[MS-WKST]:
Workstation Service Remote Protocol Specification

Revision Summary

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

Date	Revision History	Revision Class	Comments
06/01/2007	1.3.1	Editorial	Revised and edited the technical content.
07/03/2007	2.0	Major	Updated and revised the technical content.
07/20/2007	2.1	Minor	Revised technical and editorial content based on feedback.
08/10/2007	3.0	Major	Updated and revised the technical content.
09/28/2007	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.
01/25/2008	3.4	Minor	Updated the technical content.

Table of Contents

1 Introduction

The Workstation Service Remote Protocol is a Microsoft-proprietary RPC-based protocol used for remotely querying and configuring certain aspects of a **Server Message Block (SMB) network redirector** on a remote **computer**. The protocol also provides functionality to manage **domain** membership and the computer names of a computer on a network.

1.1 Glossary

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

Anonymous Session
ASCII
Built-in Domain
Client
Computer
Computer Name
Distinguished Name (DN)
DNS Name
Domain
Domain Admins
Domain Controller (DC)
Domain Object
Domain User
Endpoint
Globally Unique Identifier (GUID)
Handle
Interface Definition Language (IDL)
LDAP
Machine
Machine Account
Microsoft Interface Definition Language (MIDL)
Naming Context (NC)
NetBIOS
Network Data Representation (NDR)
Network Redirector
Opnum
Original Equipment Manufacturer (OEM) Character Set
Remote Procedure Call (RPC)
Read-Only Domain Controller (RODC)
RPC Transport
Salt
Security Context
Server
Server Message Block (SMB)
Service Principal Name (SPN)
Stock Keeping Unit (SKU)
Unicode
Universally Unique Identifier (UUID)
Well-Known Endpoint

The following terms are specific to this document:

Active User: A user that is currently authenticated on a **computer**.

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

Client Side: The initiating end of the protocol.

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 being guessed as a true random source.

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 an **RPC** protocol, a network layer protocol, and a transport layer protocol. For example, the protocol sequence NCACN_IP_TCP describes a Network Computing Architecture (NCA) connection over the Internet Protocol (IP) with a Transmission Control Protocol (TCP) as transport. For more information, see [C706] and [MS-RPCE].

Server Side: The receiving end of the protocol.

SMB Connection: A raw transport connection between an **SMB client** and **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 supported by both the **SMB client** and the **SMB server**. An **SMB Connection** is sometimes referred to as a virtual connection, as specified in [CIFS].

SMB Session: An authenticated user connection established between an **SMB client** and **SMB server** over an **SMB Connection**.

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

1.2 References

1.2.1 Normative References

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

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

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

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

[FIPS186-2] National Institute of Standards and Technology, "Federal Information Processing Standards Publication 186-2: Digital Signature Standard (DSS)", January 2000, http://csrc.nist.gov/publications/fips/fips186-2/fips186-2-change1.pdf

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

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

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

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

[MS-DRSR] Microsoft Corporation, "Directory Replication Service (DRS) Remote Protocol Specification", July 2007.

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

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

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

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

[MS-LSAT] Microsoft Corporation, "Local Security Authority (Translation Methods) Remote Protocol Specification", June 2007.

[MS-NLMP] Microsoft Corporation, "NT LAN Manager (NTLM) Authentication Protocol Specification", June 2007.

[MS-NRPC] Microsoft Corporation, "Netlogon Remote Protocol Specification", March 2007.

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

[MS-SAMR] Microsoft Corporation, "Security Account Manager (SAM) Remote Protocol Specification (Client-to-Server)", June 2007.

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

[MS-SMB2] Microsoft Corporation, "Server Message Block (SMB) Version 2.0 Protocol Specification", July 2007.

[MS-SPNG] Microsoft Corporation, "Simple and Protected Generic Security Service Application Program Interface Negotiation Mechanism (SPNEGO) Protocol Extensions", January 2007.

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

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

[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, R., "Domain Names - Implementation and Specification", 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

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

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

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

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

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

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

[RFC2254] Howes, T., "The String Representation of LDAP Search Filters", RFC 2254, December 1997, http://www.ietf.org/rfc/rfc2254.txt

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

[RFC4086] Eastlake III, D., Schiller, J., and Crokcer, S., "Randomness Requirements for Security", 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.

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

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

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

1.2.2 Informative References

[FIPS140] National Institute of Standards and Technology, "Federal Information Processing Standards Publication 140-2: Security Requirements for Cryptographic Modules", December 2002, http://csrc.nist.gov/publications/fips/fips140-2/fips1402.pdf

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

[SYSERR] Microsoft Corporation, "System Error Codes", http://msdn2.microsoft.com/en-us/library/ms681381.aspx

[WININTERNALS] Russinovich, M. and Solomon, D., "Microsoft Windows Internals, Fourth Edition", Microsoft Press, 2005, ISBN: 0735619174.http://www.microsoft.com/MSPress/books/6710.aspx

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

1.3 Protocol Overview (Synopsis)

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, you can use this protocol to query the computer name or major and minor version numbers of the operating system running on a remote computer.

You can also use the protocol to configure the behavior of an SMB network redirector. For example, you 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 will be enumerated. The protocol also does not guarantee that the enumerated users or transport protocols will not be 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 **server** state and state at a **domain controller** during message processing; state which 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 Relationship to Other Protocols

This protocol is dependent upon **RPC** and SMB for its transport. This protocol uses RPC over named pipes, as specified in section 2.1. Named pipes in turn use the SMB protocol, as specified in [MS-SMB]. <1> <2>

This protocol modifies the state *domain-secret* which [MS-NRPC] depends on, as specified in section 3.2.1.2.

This protocol depends on the state maintained in [MS-LSAD], as specified in section 3.2.1.2.

No other protocol currently 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 invoked.

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 Windows error number space specified in [MS-ERREF]. Vendors SHOULD<3> reuse those values with their indicated meaning. 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

The following sections specify how Workstation Service Remote Protocol messages are encapsulated on the wire, and common Workstation Service Remote Protocol data types.

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** MUST be 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 2.2.1.

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 the second bullet of section 3.3.3.4.3 of [MS-RPCE]. The server SHOULD <4> use this identity to perform method-specific access checks as specified in section 3.2.4.

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 Data Types

2.2.1.1 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 for the RPC connection (that refers to this **handle**) to IDENTIFICATION. IDENTIFICATION implies an impersonation level of SECURITY_IDENTIFICATION. For more information on impersonation levels, see [MS-RPCE] section 2.2.1.1.9.

2.2.1.2 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 for the RPC connection (that refers to this handle) to IMPERSONATION. IMPERSONATION implies an impersonation level of SECURITY_IMPERSONATION. For more information on impersonation levels, see [MS-RPCE] section 2.2.1.1.9.

2.2.1.3 handle_t

A concrete type for a RPC binding handle, as specified in [C706] section 4.2.9.7 and [MS-DTYP] section 2.1. The client MUST set the impersonation level for the RPC connection (that refers to this handle) to IMPERSONATION. IMPERSONATION implies an impersonation level of SECURITY_IMPERSONATION. For more information on impersonation levels, see [MS-RPCE] section 2.2.1.1.9.

2.2.1.4 MAX_PREFERRED_LENGTH

```
#define MAX_PREFERRED_LENGTH   -1
```

2.2.1.5 MAXULONG

```
#define MAXULONG    0xffffffff
```

2.2.2 Enumerations

2.2.2.1 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.

NetSetupWorkgroupName: Machine is joined to a workgroup.

NetSetupDomainName: Machine is joined to a domain.

2.2.2.2 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.

2.2.2.3 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 3.2.4.22)** 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.

2.2.3 Unions

2.2.3.1 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 2.2.4.1.

WkstaInfo101: Contains information about a computer environment. For details, see section 2.2.4.2.

WkstaInfo102: Contains information about a computer environment. For details, see section 2.2.4.3.

WkstaInfo502: Contains information about a computer environment. For details, see section 2.2.4.4.

WkstaInfo1013: Contains information about the state of the SMB network redirector. For details, see section 2.2.4.5

WkstaInfo1018: Contains information about the state of the SMB network redirector. For details, see section 2.2.4.6.

WkstaInfo1046: Contains information about the state of the SMB network redirector. For details, see section 2.2.4.7.

2.2.4 Structures

2.2.4.1 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, fully qualified DNS (as specified in [RFC1035]) or NetBIOS (as specified in [RFC1001]) name of the local computer.

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

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

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

2.2.4.2 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: MUST be the same as wki100_platform_id (see section 2.2.4.1).

wki101_computername: Must be a null-terminated, fully qualified DNS or NetBIOS name of the local computer.

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

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

wki101_ver_minor: MUST be 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.

2.2.4.3 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 MUST be the same as those for wki100_platform_id (see section 2.2.4.1).

wki102_computername: MUST be a null-terminated, fully qualified DNS or NetBIOS name of the local computer.

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

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

wki102_ver_minor: MUST be 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: MUST specify the number of users who are currently active on the computer.

2.2.4.4 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: Unused. May contain any value on transmission and MUST be ignored on receipt.

wki502_collection_time: Unused. May contain any value on transmission and MUST be ignored on receipt.

wki502_maximum_collection_count: Unused. May contain any value on transmission and MUST be ignored on receipt

wki502_keep_conn: MUST be 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: MUST be the number of seconds the server MUST wait before disconnecting an inactive session.

wki502_siz_char_buf: Unused. May contain any value on transmission and MUST be ignored on receipt.

wki502_max_threads: Unused. May contain any value on transmission and MUST be ignored on receipt.

wki502_lock_quota: Unused. May contain any value on transmission and MUST be ignored on receipt.

wki502_lock_increment: Unused. May contain any value on transmission and MUST be ignored on receipt.

wki502_lock_maximum: Unused. May contain any value on transmission and MUST be ignored on receipt.

wki502_pipe_increment: Unused. May contain any value on transmission and MUST be ignored on receipt.

wki502_pipe_maximum: Unused. May contain any value on transmission and MUST be ignored on receipt.

wki502_cache_file_timeout: Unused. May contain any value on transmission and MUST be ignored on receipt.

wki502_dormant_file_limit: MUST be 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: Unused. May contain any value on transmission and MUST be ignored on receipt.

wki502_num_mailslot_buffers: Unused. May contain any value on transmission and MUST be ignored on receipt.

wki502_num_srv_announce_buffers: Unused. May contain any value on transmission and MUST be ignored on receipt.

wki502_max_illegal_datagram_events: Unused. May contain any value on transmission and MUST be ignored on receipt.

wki502_illegal_datagram_event_reset_frequency: Unused. May contain any value on transmission and MUST be ignored on receipt.

wki502_log_election_packets: Unused. May contain any value on transmission and MUST be ignored on receipt.

wki502_use_opportunistic_locking: Unused. May contain any value on transmission and MUST be ignored on receipt.

wki502_use_unlock_behind: Unused. May contain any value on transmission and MUST be ignored on receipt.

wki502_use_close_behind: Unused. May contain any value on transmission and MUST be ignored on receipt.

wki502_buf_named_pipes: Unused. May contain any value on transmission and MUST be ignored on receipt.

wki502_use_lock_read_unlock: Unused. May contain any value on transmission and MUST be ignored on receipt.

wki502_utilize_nt_caching: Unused. May contain any value on transmission and MUST be ignored on receipt.

wki502_use_raw_read: Unused. May contain any value on transmission and MUST be ignored on receipt.

wki502_use_raw_write: Unused. May contain any value on transmission and MUST be ignored on receipt.

wki502_use_write_raw_data: Unused. May contain any value on transmission and MUST be ignored on receipt.

wki502_use_encryption: Unused. May contain any value on transmission and MUST be ignored on receipt.

wki502_buf_files_deny_write: Unused. May contain any value on transmission and MUST be ignored on receipt.

wki502_buf_read_only_files: Unused. May contain any value on transmission and MUST be ignored on receipt.

wki502_force_core_create_mode: Unused. May contain any value on transmission and MUST be ignored on receipt.

wki502_use_512_byte_max_transfer: Unused. May contain any value on transmission and MUST be ignored on receipt.

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

2.2.4.5 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.

2.2.4.6 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.

2.2.4.7 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.

2.2.4.8 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;
  int wkti0_wan_ish;
} WKSTA_TRANSPORT_INFO_0,
 *PWKSTA_TRANSPORT_INFO_0,
```

```
*LPWKSTA_TRANSPORT_INFO_0;
```

wkti0_quality_of_service: Unused. May contain any value on transmission and MUST be ignored on receipt.

wkti0_number_of_vcs: MUST be the current number of remote connections using this transport protocol.

wkti0_transport_name: MUST be the null-terminated, implementation-specific<5> name of the transport protocol.

wkti0_transport_address: MUST be 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.

2.2.4.9 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 may be currently active on a computer; this MUST be the name of any such user.

2.2.4.10 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, fully qualified DNS names of other domains of which the computer is aware. The domain names MUST be separated by a space.

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

2.2.4.11 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: MUST specify 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: MUST specify the total number of bytes the SMB network redirector received.

SmbsReceived: MUST specify the total number of SMB messages that the SMB network redirector received.

PagingReadBytesRequested: The interpretation of this field is server implementation-specific. See section 3.2.4.7.

NonPagingReadBytesRequested: The interpretation of this field is server implementation-specific. See section 3.2.4.7.

CacheReadBytesRequested: The interpretation of this field is server implementation-specific. See section 3.2.4.7.

NetworkReadBytesRequested: The interpretation of this field is server implementation-specific. See section 3.2.4.7.

BytesTransmitted: MUST specify the total number of bytes that the SMB network redirector transmitted.

SmbsTransmitted: MUST specify the total number of SMB messages that the SMB network redirector transmitted.

PagingWriteBytesRequested: The interpretation of this field is server implementation-specific. See section 3.2.4.7.

NonPagingWriteBytesRequested: The interpretation of this field is server implementation-specific. See section 3.2.4.7.

CacheWriteBytesRequested: The interpretation of this field is server implementation-specific. See section 3.2.4.7.

NetworkWriteBytesRequested: The interpretation of this field is server implementation-specific. See section 3.2.4.7.

InitiallyFailedOperations: MUST specify the total number of network operations that failed to start.

FailedCompletionOperations: MUST specify the total number of network operations that failed to complete.

ReadOperations: MUST specify the total number of read operations that the SMB network redirector initiated.

RandomReadOperations: The interpretation of this field is server implementation-specific. See section 3.2.4.7..

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

LargeReadSmbs: MUST specify 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, as specified in [MS-SMB].

SmallReadSmbs: MUST specify 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: MUST specify the total number of write operations that the SMB network redirector initiated.

RandomWriteOperations: The interpretation of this field is server implementation-specific. See section 3.2.4.7.

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

LargeWriteSmbs: MUST specify 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, as specified in [MS-SMB], section 2.2.12.17 .

SmallWriteSmbs: MUST specify 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-SMB], section 2.2.12.17 .

RawReadsDenied: MAY<8> specify the total number of raw read requests made by the SMB network redirector that have been denied by the remote computer, as specified in [MS-SMB].

RawWritesDenied: MAY<9> specify the total number of raw write requests made by the SMB network redirector that have been denied by the remote computer, as specified in [MS-SMB].

NetworkErrors: MUST specify the total number of network errors that the SMB network redirector received.

Sessions: MUST specify the total number of remote SMB sessions that the SMB network redirector established.

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

Reconnects: MUST specify the total number of SMB connections that have failed.

CoreConnects: MUST specify the total number of SMB connections to remote computers supporting the PCNET1 dialect that have succeeded, as specified in [MS-SMB], section 3.2.4.2.2.

Lanman20Connects: MUST specify the total number of SMB connections that have succeeded to remote computers supporting the LM1.2X002 dialect, as specified in [MS-SMB], section 3.2.4.2.2.

Lanman21Connects: MUST specify the total number of SMB connections that have succeeded to remote computers supporting the LANMAN2.1 dialect, as specified in [MS-SMB], section 3.2.4.2.2.

LanmanNtConnects: MUST specify the total number of SMB connections that have succeeded to remote computers supporting the NTLANMAN dialect, as specified in [MS-SMB], section 3.2.4.2.2.

ServerDisconnects: MUST specify the number of times that a remote computer disconnected the SMB network redirector.

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

UseCount: MUST specify the total number of SMB connections that the SMB network redirector established.

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

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

2.2.4.12 WKSTA_USER_INFO_0_CONTAINER

The **WKSTA_USER_INFO_0_CONTAINER** structure contains a value that indicates the number of entries that the **NetrWkstaUserEnum** 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: MUST be the number of entries that the method returned.

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

2.2.4.13 WKSTA_USER_INFO_1_CONTAINER

The **WKSTA_USER_INFO_1_CONTAINER** structure contains a value that indicates the number of entries that the **NetrWkstaUserEnum** 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: MUST be the number of entries that the method returned.

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

2.2.4.14 WKSTA_USER_ENUM_STRUCT

The **WKSTA_USER_ENUM_STRUCT** structure is used by the **NetrWkstaUserEnum** 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 **WKSTA_USER_INFO_0_CONTAINER** type.
0x00000001	Specifies the **WKSTA_USER_INFO_1_CONTAINER** type.

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

2.2.4.15 WKSTA_TRANSPORT_INFO_0_CONTAINER

The **WKSTA_TRANSPORT_INFO_0_CONTAINER** structure is used by the **NetrWkstaTransportEnum** method. This structure holds a value that specifies the number of entries and a pointer to the base structure type **WKSTA_TRANSPORT_INFO_0** 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.

2.2.4.16 WKSTA_TRANSPORT_ENUM_STRUCT

The **WKSTA_TRANSPORT_ENUM_STRUCT** structure is used by the **NetrWkstaTransportEnum** 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 0x00000000.

WkstaTransportInfo: Contains a pointer to a **WKSTA_TRANSPORT_INFO_0_CONTAINER** structure.

2.2.4.17 JOINPR_USER_PASSWORD

The **JOINPR_USER_PASSWORD** structure includes a buffer that contains an unencrypted **salt**, a buffer that contains the **cleartext** password padded with random bytes (as needed), and the count of bytes in the cleartext password buffer. This structure is defined here as a reference point to use in the next section for explaining the layout of memory in the JOINPR_ENCRYPTED_USER_PASSWORD.Buffer field.

JOIN_OBFUSCATOR_LENGTH MUST be 8.

JOIN_MAX_PASSWORD_LENGTH MUST be 256.

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

Obfuscator: An unsigned character that contains a value (a salt) that MUST be filled with random bytes by the requester.

Buffer: This array of characters, in little-endian order, MUST contain the cleartext value at the end of the buffer. The cleartext value can be no more than 512 bytes. The start of the string MUST be **Length** number of bytes from the end of the buffer. The unused portions of JOINPR_USER_PASSWORD.Buffer SHOULD be filled with random bytes by the requester.

Length: An unsigned integer, in little-endian order, that MUST specify the number of bytes of the cleartext value (located in JOINPR_USER_PASSWORD.Buffer).

2.2.4.18 JOINPR_ENCRYPTED_USER_PASSWORD

The **JOINPR_ENCRYPTED_USER_PASSWORD** structure includes a buffer laid out in the format of a **JOINPR_USER_PASSWORD (section 2.2.4.17)** structure. The buffer contains an unencrypted

salt (the **Obfuscator** field specified in section 2.2.4.17), followed by an encrypted portion that contains random bits for padding (as needed) to a fixed width for obfuscation, followed by an encoded password that is prefixed with a value used to seed a run-encoding algorithm (the **Buffer** field as specified in section 2.2.4.17) followed by the length of the encoded password, specified as a 4-byte integer (the **Length** field in section 2.2.4.17).

JOIN_OBFUSCATOR_LENGTH MUST be equal to 8.

JOIN_MAX_PASSWORD_LENGTH MUST be equal to 256.

```
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 524 bytes. The decrypted format of **Buffer** is the **JOINPR_USER_PASSWORD** structure. The explanation that follows refers to that structure definition to explain how these bytes are initialized by the caller and interpreted by the server during message processing.

The following is a detailed explanation of the encoding and encryption process, followed by a similar description of the decryption and decoding. The encoding and encryption process is performed by the caller, but is included for completeness and to facilitate the reader's understanding of server message processing. The server simply decrypts and decodes **Buffer** to extract the cleartext password.

The description covers the following distinct steps:

1. Encoding the cleartext password.
2. Initializing a **JOINPR_USER_PASSWORD** structure using the result from step 1.
3. Initializing a **JOINPR_ENCRYPTED_USER_PASSWORD.Buffer** with the encrypted value of step 2. The same algorithm decrypts **JOINPR_ENCRYPTED_USER_PASSWORD.Buffer**.
4. Decoding the result of decryption, interpreted as a **JOINPR_USER_PASSWORD** structure, to recover the cleartext password.

2.2.4.18.1 Password Encoding

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

First, the cleartext password represented as a Unicode string is encoded using the following sequence:

PasswordLength: The number of characters in the cleartext password.

EncodedPassword: A buffer of length ((*PasswordLength* + 1) * 2) bytes.

Seed: A single byte.

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

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

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

Seed and 0x43 MUST be combined with a bitwise OR operator. The first byte (index 0) of the buffer *EncodedPassword* MUST be equal to the value of combining the first byte and the result of the previous bitwise OR operation with a bitwise XOR operator.

For each subsequent byte I, beginning at index 1, 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*.

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 above algorithm:

- *PasswordLength* is the number of characters in the cleartext password.
- *EncodedPassword* is a zero-initialized buffer of ((*PasswordLength* + 1) * 2) bytes.
- Copy the cleartext password into *EncodedPassword* beginning at the third byte (zero-based index of 2).

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 1: EncodedPassword buffer

where the double byte element with index 0 contains 16 bits all set to zero.

The *Seed* is set to a non-zero value chosen at random.

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
0xAB	0x00	0xBB	0x10	0xFA	0x51	0xA9	0x02	0xFA	0x51	0xAD	0x06	0xE2	0x49	0xB0	0x1B	0xF4	0x5F	0x00	0x00

Figure 2: EncodedPassword buffer

The first byte of the buffer *EncodedPassword* is equal to the value of *Seed* and 0x43 combined using a bitwise OR operatation.

Each subsequent byte I, beginning at index 1, equals 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*.

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

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

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 BB 00 41 00 53 00 53 00 57 00 4F 00 52 00 44 00
00 00 BB 10 41 00 53 00 53 00 57 00 4F 00 52 00 44 00
00 00 BB 10 FA 00 53 00 53 00 57 00 4F 00 52 00 44 00
00 00 BB 10 FA 51 53 00 53 00 57 00 4F 00 52 00 44 00
00 00 BB 10 FA 51 A9 00 53 00 57 00 4F 00 52 00 44 00
00 00 BB 10 FA 51 A9 02 53 00 57 00 4F 00 52 00 44 00
00 00 BB 10 FA 51 A9 02 FA 00 57 00 4F 00 52 00 44 00
00 00 BB 10 FA 51 A9 02 FA 51 57 00 4F 00 52 00 44 00
00 00 BB 10 FA 51 A9 02 FA 51 AD 00 4F 00 52 00 44 00
00 00 BB 10 FA 51 A9 02 FA 51 AD 06 4F 00 52 00 44 00
00 00 BB 10 FA 51 A9 02 FA 51 AD 06 E2 00 52 00 44 00
00 00 BB 10 FA 51 A9 02 FA 51 AD 06 E2 49 52 00 44 00
00 00 BB 10 FA 51 A9 02 FA 51 AD 06 E2 49 B0 00 44 00
00 00 BB 10 FA 51 A9 02 FA 51 AD 06 E2 49 B0 1B 44 00
00 00 BB 10 FA 51 A9 02 FA 51 AD 06 E2 49 B0 1B f4 00
00 00 BB 10 FA 51 A9 02 FA 51 AD 06 E2 49 B0 1B f4 5F
```

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
```

The encoding is complete. Assuming *Seed* was given a value of 171 decimal the example buffer would look like the one below:

0	1	2	3	4	5	6	7	8	9	10	1	2	3	4	5	6	7	8	9
0x00	0xAB	0x10	0xBB	0x71	0xDA	0x02	0xA9	0x71	0xDA	0x06	0xAD	0x69	0xC2	0x1B	0xB0	0x7F	0xD4	0x00	0x00

Figure 3: EncodedPassword complete

2.2.4.18.2 Initialize a JOINPR_USER_PASSWORD Structure

Next, the *EncodedPassword* MUST be packed into a **JOINPR_USER_PASSWORD** structure as follows.

- **JOINPR_USER_PASSWORD.Obfuscator** is initialized with 8 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** MUST specify the number of bytes in *EncodedPassword*.

2.2.4.18.3 Encryption and Decryption

The encryption algorithm that encrypts **JOINPR_USER_PASSWORD** beginning at **JOINPR_USER_PASSWORD.Buffer** and including **JOINPR_USER_PASSWORD.Length** MUST be specified by the following pseudo-code. **JOINPR_USER_PASSWORD.Obfuscator** MUST not be encrypted as it salts the shared secret session key used for encryption and decryption as described below.

MD5Init, MD5Update, and MD5Final are predicates/functions specified in [RFC1321]. md5Context is a variable of type MD5_CTX, as specified in [RFC1321]. rc4_key and rc4 are functions/predicates specified in [SCHNEIER]. rc4key is a variable of type RC4_KEYSTRUCT, as specified in [SCHNEIER]. encrypted-buffer MUST be **JOINPR_USER_PASSWORD.Buffer** and **JOINPR_USER_PASSWORD.Length**, which is 516 bytes. user-session-key MUST be a 16-byte value obtained from the 16-byte SMB ([MS-SMB]) session key.

```
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)
```

JOINPR_ENCRYPTED_USER_PASSWORD.Buffer is initialized with the contents of **JOINPR_USER_PASSWORD**.

2.2.4.18.4 Password Decoding

Decrypt the encrypted portion of the structure **JOINPR_USER_PASSWORD (section 2.2.4.17)**.

You MUST use the following algorithm to encode the password. However, you 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] MUST be 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] MUST be 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.

2.2.4.19 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: MUST be 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: MUST be the total size, in bytes, of memory allocated for the **Buffer**. Up to **MaximumLength** bytes can be written into the **Buffer** without overwriting memory. This value SHOULD be a multiple of 2. If not, the server MUST decrement this value by 1. This value MUST not be less than **Length**.

Buffer: Contains the actual Unicode UTF-8 string. If the **MaximumLength** field is greater than zero, then this field MUST contain a non-NULL value. **Buffer** may or may not be NULL-terminated

2.2.4.20 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: MUST be the number of entries that the method call returns.

ComputerNames: MUST specify the names as an array of **UNICODE_STRINGS** that are associated with a machine.

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 below, a **server-side** implementation of this protocol may 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 document does not mandate that implementations adhere to this model as long as their external behavior is consistent with that described in this document.

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 may 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 may be logical members of a domain; in that case, associated with each logical user is the domain of which the user is a member.

3.2.1.1 Computer Name Abstract Data Model

The following persisted variables are part of the state of any machine on a network and MUST be available to other protocols on the same machine::

computer-name: A tuple containing:

- **netbios:** The NetBIOS name of the computer
- **dns:** The fully qualified DNS name of the computer

If a computer has a DNS name, it is contained in **computer-name.dns**. If a computer has a NetBIOS name, it is contained in **computer-name.netbios**. The value of at least one of these variables MUST NOT be NULL.

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

- **netbios:** An alternate NetBIOS name of the computer
- **dns:** An alternate fully qualified DNS name of the computer

The list of alternate-computer-names MAY be empty.<10>

3.2.1.2 Domain Membership Abstract Data Model

The server MUST have read/write access to the database described as Primary Domain Information (as specified in [MS-LSAD] section 3.1.1.1) on that machine. In this document, the domain name in this database is referenced as **PrimaryDomain.DomainName**, and the security identifier (SID) in this database is referenced as **PrimaryDomain.Sid**. Any protocol depending on knowledge of domain membership change SHOULD monitor the **PrimaryDomain.Sid** for changes.

The server MUST have read/write access to the *domain-secret*, as specified in [MS-ADTS] section 7.4.1.

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.

3.2.4 Message Processing Events and Sequencing Rules

This protocol requires that the RPC runtime MUST perform a strict **network data representation (NDR)** data consistency check at target level 6.0, as specified in [MS-RPCE] section 3.

This protocol requires that the RPC runtime MUST reject a NULL unique or full pointer with a non-zero conformant value, as defined in [MS-RPCE] section 3.

Methods that accept a **WKSSVC_IDENTIFY_HANDLE** MUST return an error to the requestor if the impersonation level for the RPC connection (that refers to this handle) is not set to **SECURITY_IDENTIFICATION**, as specified in [MS-RPCE] section 2.2.1.1.9.

Methods that accept a **WKSSVC_IMPERSONATE_HANDLE** or handle_t MUST return an error to the requestor if the impersonation level for the RPC connection (that refers to this handle) is not set to SECURITY_IDENTIFICATION, as specified in [MS-RPCE] section 2.2.1.1.9.

The server SHOULD <11> enforce appropriate security measures to make sure that the caller has the required permissions to execute the following routines. For information on how to determine the identity of the caller for the purpose of performing an access check, see [MS-RPCE] section 3.3.3.1.5.

The methods MUST NOT throw an exception.

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

Methods in RPC Opnum Order

Method	Description
NetrWkstaGetInfo	Returns information about the configuration of a workstation. Opnum: 0
NetrWkstaSetInfo	Configures the permanent settings for a workstation. Opnum: 1
NetrWkstaUserEnum	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

Method	Description
NetrWkstaTransportEnum	Returns information about the settings of the network redirector. Opnum: 5
NetrWkstaTransportAdd	Binds the transport to the network redirector. Opnum: 6
NetrWkstaTransportDel	Unbinds the transport from the network redirector. Opnum: 7
Opnum8NotUsedOnWire	Reserved for local use. Opnum: 8
Opnum9NotUsedOnWire	Reserved for local use. Opnum: 9
Opnum10NotUsedOnWire	Reserved for local use. Opnum: 10
Opnum11NotUsedOnWire	Reserved for local use. Opnum: 11
Opnum12NotUsedOnWire	Reserved for local use. Opnum: 12
NetrWorkstationStatisticsGet	Retrieves workstation statistics. Opnum: 13
Opnum14NotUsedOnWire	Reserved for local use. Opnum: 14
Opnum15NotUsedOnWire	Reserved for local use. Opnum: 15
NetrJoinDomain	Joins a computer to a workgroup or domain. Opnum: 16
NetrUnjoinDomain	Unjoins a computer from a workgroup or domain. Opnum: 17
NetrRenameMachineInDomain	Changes the name of a computer in a domain. Opnum: 18
NetrValidateName	Verifies the validity of a computer, workgroup, or domain name. Opnum: 19
NetrGetJoinInformation	Retrieves join-status information for a specified computer. Opnum: 20
NetrGetJoinableOUs	Retrieves a list of organizational units (OUs) for account creation. Opnum: 21

Method	Description
NetrJoinDomain2	Uses encrypted credentials to join a computer to a workgroup or domain. Opnum: 22
NetrUnjoinDomain2	Uses encrypted credentials to unjoin a computer from a workgroup or domain. Opnum: 23
NetrRenameMachineInDomain2	Uses encrypted credentials to rename a computer in a domain. Opnum: 24
NetrValidateName2	Uses encrypted credentials to verify the validity of a computer, workgroup, or domain name. Opnum: 25
NetrGetJoinableOUs2	Uses encrypted credentials to retrieve a list of OUs for account creation. Opnum: 26
NetrAddAlternateComputerName	Adds an alternate name for a specified server. Opnum: 27
NetrRemoveAlternateComputerName	Removes an alternate name for a specified server. Opnum: 28
NetrSetPrimaryComputerName	Sets the primary computer name for a specified server. Opnum: 29
NetrEnumerateComputerNames	Returns a list of computer names for a specified server. Opnum: 30

In the table above, the term "Reserved for local use" means that the client MUST NOT send the **opnum**, and the server behavior is undefined <12> since it does not affect interoperability.

All methods MUST NOT throw an exception.

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: Pointer to a Unicode string that MUST specify the server. The client MUST map this string to an RPC binding handle, and the server MUST ignore this argument. For more information, see [C706] sections 4.3.5 and 5.1.5.2.

Level: Specifies 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. Decimal value is 100.
0x00000065	Information to be returned is of type **WKSTA_INFO_101** structure. Decimal value is 101.
0x00000066	Information to be returned is of type **WKSTA_INFO_102** structure. Decimal value is 102.
0x000001F6	Information to be returned is of type **WKSTA_INFO_502** structure. Decimal value is 502.

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

Return Values: The method MUST return the following error codes for the specified conditions:

Return value/code	Description
0x00000000 ERROR_SUCCESS	Success.
0x00000005 ERROR_ACCESS_DENIED	Access is denied.

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.

The server SHOULD<13> enforce appropriate security measures to make sure 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. For more information on how to determine the identity of the caller for the purpose of performing an access check, see [MS-RPCE] section 3.3.3.1.5.

If the *Level* parameter equals 0x00000064, then the server MUST fill in the **WkstaInfo100** member of the *WkstaInfo* parameter, as specified in section 2.2.4.1.

If the *Level* parameter equals 0x00000065, then the server MUST fill in the **WkstaInfo101** member of the *WkstaInfo* parameter, as specified in section 2.2.4.2.

If the *Level* parameter equals 0x00000066, then the server MUST fill in the **WkstaInfo102** member of the *WkstaInfo* parameter, as specified in section 2.2.4.3.

If the *Level* parameter equals 0x000001F6, then the server MUST fill in the **WkstaInfo502** member of the *WkstaInfo* parameter, as specified in section 2.2.4.4.

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: Pointer to a Unicode string specifying the server. The client MUST map this string to an RPC binding handle, and the server MUST ignore this argument. For more information, see [C706] sections 4.3.5 and 5.1.5.2.

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

Value	Meaning
0x000001F6	The *WkstaInfo* parameter points to a **WKSTA_INFO_502** structure that contains information about the computer environment. Decimal value is 502.
0x000003F5	The *WkstaInfo* parameter points to a **WKSTA_INFO_1013** structure. Decimal value is 1013.
0x000003FA	The *WkstaInfo* parameter points to a **WKSTA_INFO_1018** structure. Decimal value is 1018.
0x00000416	The *WkstaInfo* parameter points to **WKSTA_INFO_1046** structure. Decimal value is 1046.

WkstaInfo: Pointer to the buffer that MUST specify the data. The format of this data depends on the value of the *Level* parameter.

ErrorParameter: An output only parameter containing a pointer to a value that, if non-NULL, MUST receive an unsigned long that corresponds to the member of the WkstaInfo structure that caused the ERROR_INVALID_PARAMETER error.

This parameter is meaningful only if the method returns ERROR_INVALID_PARAMETER and *Level* is equal to one of the values specified in the above table.

Return Values: The method MUST return the following error codes for the specified conditions:

Return value/code	Description
0x00000000 ERROR_SUCCESS	Success.
0x00000057 ERROR_INVALID_PARAMETER	One of the function parameters is not valid.
0x00000005 ERROR_ACCESS_DENIED	Access is denied.

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

On receiving the **NetrWkstaSetInfo** method, if the *Level* parameter does not equal one of the valid values, then the server MUST fail the call. If the *Level* is 0x000003F2-0x000003F4, 0x000003FF,

0x00000409, 0x00000411-0x00000415, or 0x00000417-0x00000426, the server MUST ignore these values and MAY<14> return ERROR_INVALID_PARAMETER. Otherwise the method MUST be processed as follows.

The server MUST update the computer setting corresponding to the *WkstaInfo* parameter as follows:

- If the *Level* parameter equals 502 (0x000001F6), then all the settings defined by **WKSTA_INFO_502** MUST be updated. The server MUST validate each field (as specified in the table below) of the **WKSTA_INFO_502** structure passed in parameter *WkstaInfo*.
- If the *Level* parameter equals 1013, 1018 or 1046, then the server MUST validate the value of the field contained within the **WKSTA_INFO_1013**, **WKSTA_INFO_1018**, or **WKSTA_INFO_1046** structure, respectively. If a specific field is found to be invalid, then the returned value for ErrorParameter MUST be set as specified in the following table.

Value	Member
0	wki502_max_cmds
13	wki1013_keep_conn
18	wki1018_sess_timeout
46	wki1046_dormant_file_limit

The valid ranges for members of a WKSTA_INFO structure MUST be as follows:

Member	Valid Range
wki502_char_wait	This field is not used. The sender MAY initialize it to any value. The receiver SHOULD ignore this field.<15>
wki502_collection_time	This field is not used. The sender MAY initialize it to any value. The receiver SHOULD ignore this field.<16>
wki502_maximum_collection_count	This field is not used. The sender MAY initialize it to any value. The receiver SHOULD ignore this field.<17>
wki502_keep_conn	1 - 65535
wki502_max_cmds	50 - 65535
wki502_sess_timeout	60 - 65535
wki502_siz_char_buf	This field is not used. The sender MAY initialize it to any value. The receiver SHOULD ignore this field.<18>
wki502_max_threads	This field is not used. The sender MAY initialize it to any value. The receiver SHOULD ignore this field.<19>
wki502_lock_quota	This field is not used. The sender MAY initialize it to any value. The receiver SHOULD ignore this field.<20>

Member	Valid Range
wki502_lock_increment	This field is not used. The sender MAY initialize it to any value. The receiver SHOULD ignore this field.<21>
wki502_lock_maximum	This field is not used. The sender MAY initialize it to any value. The receiver SHOULD ignore this field.<22>
wki502_pipe_increment	This field is not used. The sender MAY initialize it to any value. The receiver SHOULD ignore this field.<23>
wki502_pipe_maximum	This field is not used. The sender MAY initialize it to any value. The receiver SHOULD ignore this field.<24>
wki502_cache_file_timeout	0 - MAXULONG
wki502_dormant_file_limit	1 - MAXULONG
wki502_read_ahead_throughput	This field is not used. The sender MAY initialize it to any value. The receiver SHOULD ignore this field.<25>
wki502_num_mailslot_buffers	This field is not used. The sender MAY initialize it to any value. The receiver SHOULD ignore this field.<26>
wki502_num_srv_announce_buffers	This field is not used. The sender MAY initialize it to any value. The receiver SHOULD ignore this field.<27>
wki502_max_illegal_datagram_events	This field is not used. The sender MAY initialize it to any value. The receiver SHOULD ignore this field.<28>
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.<29>
wki502_log_election_packets	This field is not used. The sender MAY initialize it to any value. The receiver SHOULD ignore this field.<30>
wki502_use_opportunistic_locking	This field is not used. The sender MAY initialize it to any value. The receiver SHOULD ignore this field.<31>
wki502_use_unlock_behind	This field is not used. The sender MAY initialize it to any value. The receiver SHOULD ignore this field.<32>
wki502_use_close_behind	This field is not used. The sender MAY initialize it to any value. The receiver SHOULD ignore this field.<33>
wki502_buf_named_pipes	This field is not used. The sender MAY initialize it to any value. The receiver SHOULD ignore this

Member	Valid Range
	field.<34>
wki502_use_lock_read_unlock	This field is not used. The sender MAY initialize it to any value. The receiver SHOULD ignore this field.<35>
wki502_utilize_nt_caching	This field is not used. The sender MAY initialize it to any value. The receiver SHOULD ignore this field.<36>
wki502_use_raw_read	This field is not used. The sender MAY initialize it to any value. The receiver SHOULD ignore this field.<37>
wki502_use_raw_write	This field is not used. The sender MAY initialize it to any value. The receiver SHOULD ignore this field.<38>
wki502_use_write_raw_data	This field is not used. The sender MAY initialize it to any value. The receiver SHOULD ignore this field.<39>
wki502_use_encryption	This field is not used. The sender MAY initialize it to any value. The receiver SHOULD ignore this field.<40>
wki502_buf_files_deny_write	This field is not used. The sender MAY initialize it to any value. The receiver SHOULD ignore this field.<41>
wki502_buf_read_only_files	This field is not used. The sender MAY initialize it to any value. The receiver SHOULD ignore this field.<42>
wki502_force_core_create_mode	This field is not used. The sender MAY initialize it to any value. The receiver SHOULD ignore this field.<43>
wki502_use_512_byte_max_transfer	This field is not used. The sender MAY initialize it to any value. The receiver SHOULD ignore this field.<44>
wki1013_keep_conn	1 - 65535
wki1018_sess_timeout	60-65535
wki1046_dormant_file_limit	1 - MAXULONG

The server SHOULD<45> enforce appropriate security measures to make sure 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 (0x00000005). For more information on how to determine the identity of the caller for the purpose of performing an access check, see [MS-RPCE] section 3.3.3.1.5.

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: Pointer to a Unicode string specifying the server. The client MUST map this string to an RPC binding handle, and the server MUST ignore this argument. For more information, see [C706] sections 4.3.5 and 5.1.5.2.

UserInfo: Pointer to the buffer to receive the data. The data MUST be returned as a **WKSTA_USER_ENUM_STRUCT** structure that contains a **Level** member that specifies the type of structure to return.

PreferredMaximumLength: MUST be the number of bytes to allocate for the return data. If the *PreferredMaximumLength* parameter equals MAX_PREFERRED_LENGTH, the server MUST allocate as much memory as is required to return all the requested data. Otherwise, if the *PreferredMaximumLength* is insufficient to hold all the entries, then the server MUST store as many entries as will fit in the *UserInfo* buffer and return ERROR_MORE_DATA (0x000000EA).

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

ResumeHandle: This is an optional parameter. If this parameter is not NULL and the method returns ERROR_MORE_DATA, then this parameter MUST receive an implementation-specific, non-zero value <46> that can be passed in subsequent calls to this method to continue with the enumeration.

If this parameter is NULL or points to 0x00000000, then the enumeration starts from the beginning of the list of the currently logged on users.

Return Values: The method MUST return the following error codes for the specified conditions:

Return value/code	Description
0x00000000 ERROR_SUCCESS	Success.
0x000000EA ERROR_MORE_DATA	More entries are available. *UserInfo* buffer was not large enough to contain all the entries.
0x00000005 ERROR_ACCESS_DENIED	Access is denied.

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

The server SHOULD<47> enforce appropriate security measures to make sure 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 (0x00000005). For more information on how to determine the identity of the caller for the purpose of performing an access check, see [MS-RPCE] section 3.3.3.1.5.

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

If the **Level** field equals 0x00000001, then the server MUST return an array of the names and domain information of users currently active on the computer. The server MUST return this information by filling the **WKSTA_USER_INFO_1_CONTAINER** in the **WkstaUserInfo** field of *UserInfo* parameter.

If the *PreferredMaximumLength* parameter equals MAX_PREFERRED_LENGTH, the server MUST allocate as much memory as is required to return all the requested data. Otherwise, if the *PreferredMaximumLength* is insufficient to hold all the entries, then the server MUST store as many entries as will fit in the *UserInfo* buffer and return ERROR_MORE_DATA (0x000000EA).

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.<48>
- 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 some implementation-specific value that will 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 ERROR_SUCCESS (0x00000000) or ERROR_MORE_DATA (0x000000EA), 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: Pointer to a Unicode string specifying the server. The client MUST map this string to an RPC binding handle, and the server MUST ignore this argument. For more information, see [C706] sections 4.3.5 and 5.1.5.2.

TransportInfo: Pointer to the buffer that receives the data. The data is returned as a **WKSTA_TRANSPORT_ENUM_STRUCT** structure, where the **Level** member MUST be set to 0x00000000.

PreferredMaximumLength: Number of bytes to allocate for the return data.

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

ResumeHandle: On input, if this parameter is either NULL or points to ERROR_SUCCESS, then it indicates that the enumeration MUST start from the beginning of the list of the currently enabled transport protocols. If it points to a non-zero value, it indicates that the enumeration MUST begin based on the value of *ResumeHandle* as discussed below.

If this parameter is not NULL and the method returns ERROR_MORE_DATA, then this parameter MUST receive an implementation-specific, non-zero value <49> that can be passed in subsequent calls to this method to continue with the enumeration.

Return Values: The method MUST return one of the following error codes for the specified conditions:

Return value/code	Description
0x00000000 ERROR_SUCCESS	Success.
0x000000EA ERROR_MORE_DATA	More entries are available. *TransportInfo* buffer was not large enough to contain all the entries.
0x00000005 ERROR_ACCESS_DENIED	Access is denied.

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 contained in the **WKSTA_TRANSPORT_ENUM_STRUCT** structure sent 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 **WKSTA_TRANSPORT_INFO_0_CONTAINER** in the **WkstaTransportInfo** field of the *TransportInfo* parameter.

If the *PreferredMaximumLength* parameter equals MAX_PREFERRED_LENGTH, the server MUST allocate the amount of memory required to return all the requested data. If the *PreferredMaximumLength* is insufficient to hold all the entries, then the server MUST store as many entries as will fit in the *TransportInfo* buffer and return ERROR_MORE_DATA (0x000000EA).

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. <50>

- If the *ResumeHandle* parameter is non-zero, 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 ERROR_MORE_DATA (0x000000EA), then the server MUST set *ResumeHandle* to some implementation-specific value that will 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 ERROR_SUCCESS (0x00000000) or ERROR_MORE_DATA (0x000000EA), then it MUST set the *TotalEntries* parameter to equal the total number of entries that could have been enumerated from the current resume position.

3.2.4.5 NetrWkstaTransportAdd (Opnum 6)

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

```
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: Pointer to a Unicode string specifying the server. The client MUST map this string to an RPC binding handle, and the server MUST ignore this argument, as specified in [C706] sections 4.3.5 and 5.1.5.2.

Level: Specifies the information level of the data. *Level* MUST be set to 0x00000000, meaning the *TransportInfo* parameter points to a **WKSTA_TRANSPORT_INFO_0** structure.

TransportInfo: MUST be a pointer to a **WKSTA_TRANSPORT_INFO_0** structure.

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

Return Values: The method MUST return one of the following error codes for the specified conditions:

Return value/code	Description
0x00000000 ERROR_SUCCESS	Success.
0x00000057 ERROR_INVALID_PARAMETER	One of the function parameters is not valid.
0x00000005 ERROR_ACCESS_DENIED	Access is denied.

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

If the *Level* parameter is not equal to 0x00000000, then the server MUST fail the call and return ERROR_INVALID_LEVEL (0x0000007C) .

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

If any of the input parameters are invalid, then the server SHOULD return ERROR_INVALID_PARAMETER (0x00000057). Otherwise it SHOULD return ERROR_SUCCESS (0x00000000).<51>

The server SHOULD<52> enforce appropriate security measures to make sure 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 SHOULD fail the call with ERROR_ACCESS_DENIED (0x00000005). For more information on how to determine the identity of the caller for the purpose of performing an access check, see [MS-RPCE] section 3.3.3.1.5.

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 (0x00000057). 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 add this protocol to its list of currently enabled transport protocols.

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** method. This method is deprecated.

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

ServerName: Pointer to a Unicode string specifying the server. The client MUST map this string to an RPC binding handle, and the server MUST ignore this argument, as specified in [C706] sections 4.3.5 and 5.1.5.2.

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

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

Value	Meaning
0x00000000	Fail the call if open handles are using the transport protocol.
0x00000001	Same as ERROR_SUCCESS; fail the call if open handles are using the transport protocol.

Value	Meaning
	Note The value of ERROR_SUCCESS is 0x00000000.
0x00000002	Forcefully close any open handles and disable the specified transport protocol from the SMB network redirector.

Return Values: The method MUST return one of the following error codes for the specified conditions:

Return value/code	Description
0x00000000 ERROR_SUCCESS	Success.
0x00000005 ERROR_ACCESS_DENIED	Access is denied.

If the *ForceLevel* parameter does not equal 0x00000000, 0x00000001, or 0x00000002, then the server MUST fail the call.

If the server does not support this method, then it SHOULD return ERROR_SUCCESS (0x00000000) if the *ForceLevel* parameter is valid.<53> If the server does support this method, it MUST be processed as follows.

The server SHOULD<54> enforce appropriate security measures to make sure 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 (0x00000005). For more information on how to determine the identity of the caller for the purpose of performing an access check, see [MS-RPCE] section 3.3.3.1.5.

If any open file or printer handles are using the transport protocol that this call is trying to disable, then the server behavior MUST depend on the value of the *ForceLevel* parameter. If the *ForceLevel* parameter is 0x00000000 or 0x00000001, then 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, then the server MUST remove this protocol from its list of currently enabled transport protocols.

3.2.4.7 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: Pointer to a Unicode string specifying the server. The client MUST map this string to an RPC binding handle, and the server MUST ignore this argument, as specified in sections 4.3.5 and 5.1.5.2 of [C706].

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

Level: Specifies the information level of the data. This value MUST be set to 0x00000000.

Options: This value MUST be set to 0x00000000.

Buffer: MUST be a pointer to a **STAT_WORKSTATION_0** structure that contains the statistical information.

Return Values: The method MUST return one of the following error codes for the specified conditions:

Return value/code	Description
0x00000000 ERROR_SUCCESS	Success.
0x00000005 ERROR_ACCESS_DENIED	Access is denied.

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

If the *Options* parameter does not equal 0x00000000, then the server MUST fail the call.

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 described in section 2.2.4.11. These fields indicate certain performance characteristics of an operating system and may not apply to all servers. If a field does not apply to the server, then it MUST set that field to zero.<55>

3.2.4.8 NetrJoinDomain (Opnum 16)

This **NetrJoinDomain** method SHOULD<56> not be supported. <57>

```
unsigned long NetrJoinDomain(
  [in, string, unique] WKSSVC_IMPERSONATE_HANDLE ServerName,
  [in, string] wchar_t* DomainName,
  [in, string, unique] wchar_t* MachineAccountOU,
  [in, string, unique] wchar_t* AccountName,
  [in, string, unique] wchar_t* Password,
  [in] unsigned long Options
);
```

Return Values: Windows implementations do not support this method. All calls to this method return ERROR_NOT_SUPPORTED (0x00000032).

All parameters to this method SHOULD<58> be ignored.

3.2.4.9 NetrUnjoinDomain (Opnum 17)

This **NetrUnjoinDomain** method SHOULD<59> not be supported. <60>

```
unsigned long NetrUnjoinDomain(
  [in, string, unique] WKSSVC_IMPERSONATE_HANDLE ServerName,
  [in, string, unique] wchar_t* AccountName,
  [in, string, unique] wchar_t* Password,
  [in] unsigned long Options
);
```

Return Values: Windows implementations do not support this method. All calls to this method return ERROR_NOT_SUPPORTED (0x00000032).

All parameters to this method SHOULD<61> be ignored.

3.2.4.10 NetrRenameMachineInDomain (Opnum 18)

This **NetrRenameMachineInDomain** method SHOULD<62> not be supported.<63>

```
unsigned long NetrRenameMachineInDomain(
  [in, string, unique] WKSSVC_IMPERSONATE_HANDLE ServerName,
  [in, string, unique] wchar_t* MachineName,
  [in, string, unique] wchar_t* AccountName,
  [in, string, unique] wchar_t* Password,
  [in] unsigned long Options
);
```

Return Values: Windows implementations do not support this method. All calls to this method return ERROR_NOT_SUPPORTED (0x00000032).

All parameters to this method SHOULD<64> be ignored.

3.2.4.11 NetrValidateName (Opnum 19)

This **NetrValidateName** method SHOULD<65> not be supported.<66>

```
unsigned long NetrValidateName(
  [in, string, unique] WKSSVC_IMPERSONATE_HANDLE ServerName,
  [in, string] wchar_t* NameToValidate,
  [in, string, unique] wchar_t* AccountName,
  [in, string, unique] wchar_t* Password,
  [in] NETSETUP_NAME_TYPE NameType
);
```

Return Values: Windows implementations do not support this method. All calls to this method return ERROR_NOT_SUPPORTED (0x00000032).

All parameters to this method SHOULD<67> be ignored.

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: Pointer to a Unicode string specifying the server. The client MUST map this string to an RPC binding handle, and the server MUST ignore this argument. For more information, see [C706] sections 4.3.5 and 5.1.5.2.

NameBuffer: 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 SHOULD<68> ignore this parameter on input.

BufferType: A pointer to an enumerated value from **NETSETUP_JOIN_STATUS** that specifies the status of a workstation.

Return Values: The method MUST return 0x00000000 (NERR_Success) on success; otherwise, the following is a summary of the return values that an implementation MUST return as specified by the following message processing. These error codes are specified in [MS-ERREF] section 3.

Return value/code	Description
0x00000005 ERROR_ACCESS_DENIED	Access is denied.
0x00000008 ERROR_NOT_ENOUGH_MEMORY	Not enough storage is available to process this command.

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

The following ordered statements describe the sequence of message-processing operations.

The server SHOULD<69> ensure that the caller used the NCACN_NP RPC protocol sequence and SHOULD <70> return RPC_S_PROTSEQ_NOT_SUPPORTED otherwise. For information on how to identify the protocol sequence used, see [MS-RPCE].

The server SHOULD<71> enforce appropriate security measures to make sure 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 SHOULD fail the call. For more information on how to determine the identity of the caller for the purpose of performing an access check, see [MS-RPCE] section 3.3.3.1.5.

The server MUST compute the response in the following way. For information about the abstract state variable PrimaryDomain, see section 3.2.1.2.

- If **PrimaryDomain.Name** is empty, then *BufferType* MUST be set to NetSetupUnjoined, and *NameBuffer* MUST be set to NULL.

- Else if **PrimaryDomain.Sid** is empty, then *BufferType* MUST be set to NetSetupWorkgroupName and *NameBuffer* MUST contain **PrimaryDomain.Name**.
- Else *BufferType* MUST be set to NetSetupDomainName and *NameBuffer* MUST contain **PrimaryDomain.Name**.

If no errors occurred the server MUST return NERR_Success.

3.2.4.13 NetrGetJoinableOUs (Opnum 21)

This **NetrGetJoinableOUs** method SHOULD<72> not be supported.<73>

```
unsigned long NetrGetJoinableOUs(
  [in, string, unique] WKSSVC_IMPERSONATE_HANDLE ServerName,
  [in, string] wchar_t* DomainName,
  [in, string, unique] wchar_t* AccountName,
  [in, string, unique] wchar_t* Password,
  [in, out] unsigned long* OUCount,
  [out, string, size_is(, *OUCount)]
    wchar_t** OUs
);
```

Return Values: Windows implementations do not support this method. All calls to this method return ERROR_NOT_SUPPORTED (0x00000032).

All parameters to this method SHOULD<74> be ignored.

3.2.4.14 NetrJoinDomain2 (Opnum 22)

The **NetrJoinDomain2** method uses encrypted credentials to join a computer to a workgroup or domain. This method replaces **NetrJoinDomain**.<75> While **NetrJoinDomain2** is only one possible implementation that makes the state changes necessary to affect a domain join it additionally provides the ability to remote the operation by specifying the *ServerName* parameter. 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* DomainName,
  [in, string, unique] wchar_t* MachineAccountOU,
  [in, string, unique] wchar_t* AccountName,
  [in, unique] PJOINPR_ENCRYPTED_USER_PASSWORD Password,
  [in] unsigned long Options
);
```

RpcBindingHandle: RPC binding handle (as specified in [C706]).

ServerName: Pointer to a Unicode string specifying the server. The server SHOULD<76> ignore this parameter. RPC client machines SHOULD <77> set this parameter to a string that resolves to an IP address of the server.

DomainName: Pointer to a string that specifies a name of 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 *DomainName*\MachineName, where *DomainName* 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 *DomainName* portion of this parameter MUST be either the NetBIOS name of the domain or the fully qualified DNS name of the domain. If the MachineName is passed it MUST be the NetBIOS name of the domain controller or the fully qualified DNS name of the domain controller. The format of *DomainName* places no constraint on the format of MachineName and vice versa; each of the following permutations are accepted: NetBIOS\NetBIOS, NetBIOS\DNS, DNS\NetBIOS, and DNS\DNS.

MachineAccountOU: Pointer to a string that MUST contain (as specified in [RFC1777]) the format name of the organizational unit 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: Pointer to a string that specifies an account name in the domain *DomainName* 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>, or

<UserName>@<FullyQualifiedDNSDomainName>

Password: Pointer to a **JOINPR_ENCRYPTED_USER_PASSWORD** structure that specifies the encrypted password to use with the *AccountName* parameter. For information on decrypting and decoding the password, see section 2.2.4.18. See sections 3.2.4.14.1 and 3.2.4.14.2 for processing of this parameter.

Options: A 32-bit bit-field specifying modifications to default message-processing behavior. The server MUST ignore any options that it does not understand.

Value	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 **computer-name** (see section 3.2.1.1) unless this behavior is altered by another option such as NETSETUP_JOIN_WITH_NEW_NAME.
NETSETUP_WIN9X_UPGRADE 0x00000010	Join operation is occurring as part of an upgrade from Windows Me, Windows 98, or Windows 95 to Windows XP, Windows 2000, or Windows NT.
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	Windows XP and later: Indicates that the *Password* parameter specifies the password for the machine joining

Value	Meaning
	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	Windows XP and later: Indicates that the **service principal name (SPN)** and the DnsHostName properties on the computer SHOULD NOT <78> be updated at this time, but instead SHOULD <79> be updated during a subsequent call to the **NetrRenameMachineInDomain2** method.
NETSETUP_JOIN_DC_ACCOUNT 0x00000200	Windows Vista and later: Indicates that the join SHOULD <80> be allowed if an existing account exists and it is a Domain Controller account. <81>
NETSETUP_JOIN_WITH_NEW_NAME 0x00000400	Windows Vista and later: Indicates that the join SHOULD <82>occur using the new computer name.
NETSETUP_INSTALL_INVOCATION 0x00040000	Indicates that the protocol method was invoked during installation of Windows.

Return Values: The method MUST return 0x00000000 (NERR_Success) on success; otherwise, the following is a summary of the return values that an implementation MUST return, as specified by the message processing below. These error codes are specified in [MS-ERREF] section 3.

Return value/code	Description
0x00000002 ERROR_FILE_NOT_FOUND	The object was not found.
0x00000005 ERROR_ACCESS_DENIED	Access is denied.
0x00000008 ERROR_NOT_ENOUGH_MEMORY	The server failed to allocate memory.
0x00000032 ERROR_NOT_SUPPORTED	The request is not supported.
0x00000056 ERROR_INVALID_PASSWORD	The specified network password is not correct.
0x00000057 ERROR_INVALID_PARAMETER	The parameter is incorrect.
0x0000052D ERROR_PASSWORD_RESTRICTION	Unable to update the password. The value provided for the new password does not meet the length, complexity, or history requirements of the domain.
0x0000052E ERROR_LOGON_FAILURE	Logon failure: unknown user name or bad password.

Return value/code	Description
0x00000534 ERROR_NONE_MAPPED	The account was not found.
0x0000054A ERROR_INVALID_DOMAIN_ROLE	The name of a domain controller was provided in the *DomainName* parameter, and validation of that domain controller failed. Validation is described in the message-processing steps for the section "Domain Join" later.
0x0000054B ERROR_NO_SUCH_DOMAIN	The specified domain either does not exist or could not be contacted.
0x000006FF RPC_S_CALL_IN_PROGRESS	A remote procedure call is already in progress <83>.
0x000008B0 NERR_UserExists	The account already exists.
0x00000A83 NERR_SetupAlreadyJoined	This computer is already joined to a domain.
0x00000A85 NERR_SetupDomainController	This computer is a domain controller and cannot be unjoined from a domain.
0x00000A87 NERR_InvalidWorkgroupName	The specified workgroup name is invalid.
0x0000092F NERR_InvalidComputer	This computer name is invalid.

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

The message processing of this section describes the behavior of either joining a domain or joining a workgroup. The message processing description is broken into several sections. Discussed first, in section 3.2.4.14.1, is the message processing common to both domain and workgroup joins. Discussed second, in section 3.2.4.14.2, are details of the complex state transition associated with domain join. Next, the domain join specific message processing is discussed in section 3.2.4.14.3 and, finally, workgroup join specific message processing is discussed in section 3.2.4.14.4.

3.2.4.14.1 Common Message Processing

The following declarative statements affect all message processing.

- The computer-name MUST be a locally persisted tuple where computer-name.netbios is a Unicode UTF-8 string of 15 characters or fewer. For more information, see section 3.2.1.1.
- Unless otherwise noted, if the server encounters an error during message processing the server SHOULD revert any state changes made, it MUST stop message processing, and it MUST return the error to the caller. <84>
- The server MUST ignore any flags set in **Options** that it does not support. <85><86><87>

The following are ordered statements describing the sequence of message processing operations.

- For the following discussion, *PasswordString* is defined as a UTF-8 string containing a password in cleartext.

- The server SHOULD <88> ensure that the caller used the **NCACN_NP** RPC protocol sequence and SHOULD <89> return **RPC_S_PROTSEQ_NOT_SUPPORTED** otherwise. For information on how to identify the protocol sequence used, see [MS-RPCE].

- The server SHOULD<90> enforce appropriate security measures to make sure that the caller's **security context** 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 SHOULD fail the call with **ERROR_ACCESS_DENIED** (0x00000005). For more information on how to determine the identity of the caller for the purpose of performing an access check, see [MS-RPCE] section 3.3.3.1.5.

- If *Password* is NULL then *PasswordString* MUST be NULL. Otherwise, the server MUST decrypt and decode the *Password*, as specified in section 2.2.4.18. *PasswordString* MUST be equal to the decrypted and decoded value. The decrypted buffer is represented as **JOINPR_USER_PASSWORD**, as specified in section 2.2.4.17. The Length field MUST be less than 513 bytes in length, otherwise message processing is stopped and the server MUST return **ERROR_INVALID_PASSWORD**.

- If the **DomainName** is NULL the server MUST stop message processing and return **ERROR_INVALID_PARAMETER**. Otherwise, message processing continues.

- The server SHOULD return **ERROR_NOT_SUPPORTED** if the server does not support processing of this message <91>.

- If the server processing the message is a domain controller, the server MUST stop message processing and return **NERR_SetupDomainController**. Otherwise, message processing continues.

- If *Options* does not have the bit **NETSETUP_JOIN_DOMAIN** set, then the server MUST continue processing this message, as specified in section 3.2.4.14.4; otherwise, the server MUST process the message as specified in section 3.2.4.14.3.

3.2.4.14.2 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, as specified in section 3.2.1.1 and in [MS-ADTS] section 7.4.

The state changes referenced above appear in the sequence of message processing steps later in this section 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 required state changes for a computer to join a domain.

The server MUST acquire the following from the domain:

- PrimaryDomain.DomainName and PrimaryDomain.Sid, as specified in section 3.2.1.2.

The server MUST persist in the machine:

- The server MUST store the values queried from PrimaryDomain.DomainName and PrimaryDomain.Sid in the local PrimaryDomain.DomainName and PrimaryDomain.Sid, as specified in section 3.2.1.2.

- The server MUST persist the *domain-secret*, as specified in [MS-ADTS] section 7.4.

The server MUST persist in the domain:

- A computer account object with all of the following **LDAP** attributes values <92>:

  The LDAP attributes, in order of appearance below, are specified in [MS-ADA3] sections 1.340, 1.221, and 1.330; [MS-ADA1] section 1.1.84; and [MS-ADA3] section 1.252. See [RFC2252] and [RFC2253] for more information about LDAP.

LDAP Attribute Name	Value
userAccountControl	The following bit is set to one USER_WORKSTATION_TRUST_ACCOUNT (0x00000080), and the following bit is set to zero USER_ACCOUNT_DISABLED, as specified in [MS-SAMR]. See section 3.2.5.14.2 userAccountControl Mapping Table in [MS-SAMR] for information on the mapping of these bits in the LDAP protocol.
samAccountName	The value of machine-account-name, as specified in [MS-ADTS] section 7.4. This is computer-name.netbios, as specified in section 3.2.1.1, suffixed with a "$" character.
unicodePwd	The value of *domain-secret*, as defined in [MS-ADTS]. Protocols that expose this attribute actually persist the NT Hash of *domain-secret*, as specified in [MS-SAMR] (for information about NT Hash).
dNSHostName	computer-name.dns as specified in section 3.2.1.1.
servicePrincipalName	Two values, as specified in the message processing sequence later in this section. The values are a DNS-based SPN and a NetBIOS-based SPN for the computer joining the domain.

3.2.4.14.3 Domain Join Specific Message Processing

DomainNameString: Unicode UTF-8 string with the same properties specified for the parameter *DomainName*.

DomainControllerString: UTF-8 string which MUST contain the name of a domain controller in the domain that the server is joining.

DomainObject: An object which MUST be in the domain database, as specified in [MS-ADTS] section 7.4.

MachineAccountOUString: UTF-8 string containing the organizational unit in the directory for the machine account.

ComputerAccountString: UTF-8 string containing the value stored in the samAccountName attribute of the computer object in the domain database.

ComputerPasswordString: **ASCII** string containing a password in cleartext.

DNSComputerNameString: UTF-8 string containing the DNS name of the computer.

Spn1: UTF-8 string containing a DNS based SPN for the computer joining the domain.

Spn2: UTF-8 string containing a NetBIOS based SPN for the computer joining the domain.

The following ordered statements describe the sequence of message-processing operations.

1. If the bit **NETSETUP_MACHINE_PWD_PASSED** is set in *Options* and the bit **NETSETUP_JOIN_UNSECURE** is not set in *Options*, the server MUST return **ERROR_INVALID_PARAMETER**. Otherwise, message processing continues.

2. If the bit NETSETUP_MACHINE_PWD_PASSED is set in *Options* and *AccountName* is NULL, the server MUST return ERROR_INVALID_PARAMETER. Otherwise, message processing continues.

3. If the bit NETSETUP_MACHINE_PWD_PASSED 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 bit NETSETUP_MACHINE_PWD_PASSED is set in *Options*, then the value of *PasswordString* MUST be copied to the value of *ComputerPasswordString* and *PasswordString* MUST be NULL.

5. If the server processing the message is already joined to a domain and *Options* does not have the bit **NETSETUP_DOMAIN_JOIN_IF_JOINED** set, the server MUST return **NERR_SetupAlreadyJoined**. Otherwise, message processing continues.

6. If *DomainNameString* contains the character "\", then *DomainNameString* MUST be truncated such that the value of *DomainNameString* is equal to the substring of *DomainName* 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.

7. If the string *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 equal to the string name of the located domain controller. <93>

8. Then, *DomainNameString* MUST be validated as a valid domain name. The validation process is specified in section 3.2.4.17 where NameType is NetSetupDomain (0x3). If this validation fails, the server MUST stop message processing and return any error specified in the above validation process.

9. If NETSETUP_MACHINE_PWD_PASSED is set in *Options*, the server MUST attempt to establish an authenticated Remote Procedure Call (RPC) session between the server and the domain controller named by the value of *DomainControllerString*. The RPC session MUST use the SMB NCACN_NP protocol sequence using the requester's security context. This security context is used by the domain controller for the purpose of performing access checks on all RPC operations targeting the domain controller over this transport. Specification of connecting to a server using SMB is documented in [MS-RPCE] section 2.1.1.2.

10. If NETSETUP_MACHINE_PWD_PASSED 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. Otherwise, if NETSETUP_MACHINE_PWD_PASSED is set in *Options* and the session failed to be established in the previous step 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.

11. If NETSETUP_MACHINE_PWD_PASSED is not set in *Options*, the server MUST establish an authenticated SMB session between the server and the domain controller named by the value of *DomainControllerString*. The security context established MUST be that of *AccountName*, and the credentials supplied during authentication are those specified in *PasswordString*. This security context is used by the domain controller for the purpose of performing access checks on all operations targeting the domain controller over this transport. If an error occurs, the server MUST stop message processing and return that error. Otherwise, message processing continues.

12. If *DomainControllerString* was initialized from *DomainName*, the server MUST locate a domain controller for the domain specified in *DomainNameString* by sending a location request to the server named in *DomainControllerString*. If an error occurs, the server MUST stop message processing and return that error. Otherwise, the response MUST contain the name of the located domain controller, as reported by the DC. That name SHOULD be compared to *DomainControllerString* to confirm that the domain controller specified in *DomainControllerString* by the caller is valid.<94>

13. If the computer-name is identical to *DomainNameString*, the server MUST return **ERROR_INVALID_DOMAINNAME**. Otherwise, message processing continues.

14. The server MUST query the domain controller for **PrimaryDomain.DomainName** and **PrimaryDomain.Sid**, as specified in [MS-LSAD].<95>

15. The server MUST store the values queried in the previous step in the local PrimaryDomain.DomainName and PrimaryDomain.Sid, as specified in section 3.2.1.2.

16. If *Options* does not have the bit NETSETUP_MACHINE_PWD_PASSED set, and if *Options* has either the bit **NETSETUP_WIN9X_UPGRADE** or the bit NETSETUP_JOIN_UNSECURE set, then *ComputerPasswordString* MUST be the first 14 characters of computer-name in lowercase.

17. If Options does not have the bit NETSETUP_MACHINE_PWD_PASSED set, and if Options has neither the bit NETSETUP_WIN9X_UPGRADE or the bit NETSETUP_JOIN_UNSECURE set, then *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 as specified in [FIPS186-2] Appendix 3.1. For more information, see [RFC4086]. <96>

18. The server MUST store the value of *ComputerPasswordString* locally for consumption by the security provider services when authenticating the computer. The stored password MUST be maintained by the Netlogon Protocol (as specified in [MS-NRPC]).

19. If the value of *MachineAccountOU* is not NULL, the value of *MachineAccountOUString* MUST equal *MachineAccountOU*. If the value of *MachineAccountOU* is NULL then *MachineAccountOUString* MUST equal the value specified by the well-known object identified by the **GUID** with value GUID_COMPUTERS_CONTAINER_W, as specified in [MS-ADTS].

20. If the [RFC1777] format name of the organizational unit (OU) where the object exists in the domain database, 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.

21. *ComputerAccountString* MUST equal the UTF-8 string consisting of computer-name suffixed with a "$" character.

22. *DNSComputerNameString* MUST equal the UTF-8 string computer-name.dns.

23. *Spn1* MUST be a UTF-8 string equal to the concatenation of "HOST/" and the value of *DNSComputerNameString*.

24. *Spn2* MUST be a UTF-8 string equal to the concatenation of "HOST/" and the value of *ComputerAccountString* .

25. If the **NETSETUP_ACCT_CREATE** bit is set in *Options*, then the server MUST create the **domain object** in the domain *DomainName* at *DomainControllerString*. Manipulation of the domain computer object state is exposed through various protocols (that is, LDAP as specified in

[RFC2252], [RFC2253], and [MS-SAMR]). If the domain object already exists in an organizational unit (OU) (as specified in [MS-ADSC] section "Class organizationalUnit") 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.

26. If the NETSETUP_ACCT_CREATE bit is not set in *Options* and the domain object does not already exist in the domain *DomainName* at DomainController, then the server MUST stop message processing and return **ERROR_NONE_MAPPED**. Otherwise, message processing continues.

27. If the NETSETUP_ACCT_CREATE bit is not set in *Options*, and either the NETSETUP_WIN9X_UPGRADE or NETSETUP_JOIN_UNSECURE bit is set in *Options*, then the server MUST send a request to the Net Logon 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 domain object in the LDAP attribute named 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].

28. The following LDAP attributes on *DomainObject* MUST be set to the values in the table. The security context provided to the LDAP protocol is AccountName and the credential is *PasswordString*. For more information about attributes and attribute names, see [MS-ADTS]. For more information about LDAP, see [RFC2252] and [RFC2253].<97>

LDAP Attribute Name	Value
userAccountControl	The following bit is set to one **USER_WORKSTATION_TRUST_ACCOUNT** (0x00000080), and the following bit is set to zero USER_ACCOUNT_DISABLED, as specified in [MS-SAMR]. See section 3.2.5.14.2 userAccountControl Mapping Table in [MS-SAMR] for information on the mapping of these bits in the LDAP protocol.
samAccountName	*ComputerAccountString*
unicodePwd	The value of *ComputerPasswordString*. Protocols that expose this attribute actually persist the NT Hash of the *ComputerPasswordString*, as specified in [MS-SAMR] (for information about NT Hash).

29. The following LDAP attributes on *DomainObject* MUST be set to the values in the table unless the **NETSETUP_DEFER_SPN_SET** bit is set in *Options*.<98>

LDAP Attribute Name	Value
dNSHostName	*DNSComputerNameString*
servicePrincipalName	Two values: *Spn1* *Spn2*

30. The server MUST configure the local Net Logon Remote Protocol (as specified in [MS-NRPC]) so that it is aware of being joined to a domain by the name of *DomainName*.

31. The server MUST configure the local Windows Time Service (as specified in [WTSREF]) so that it is aware of being joined to a domain.

32. The server SHOULD store the value dns-name locally so that the Domain Naming System Service registers name records for the local computer. <99>

33. The server SHOULD add the **Domain Admins** group to the local Administrators group, and the server SHOULD add the **Domain Users** group to the local users groups, as specified in [MS-SAMR]. <100>

If no errors occur, the server MUST return NERR_Success.

3.2.4.14.4 Workgroup Join Specific Message Processing

The following ordered statements describe the sequence of message processing operations.

- If the server processing the message is already joined to a domain, the server MUST return NERR_SetupAlreadyJoined. Otherwise, message processing continues.
- The *DomainName* parameter MUST be validated as a valid workgroup name. The validation process is specified in section 3.2.4.17 where NameType is NetSetupWorkgroup (0x2). If this validation fails, the server MUST return the error specified in the above validation process.
- The server MUST persist the value *DomainName* in PrimaryDomain.*DomainName* and set PrimaryDomain.Sid equal to NULL as specified in [MS-LSAD].
- The server MUST configure the local Net Logon Remote Protocol (as specified in [MS-NRPC]) so that it is aware of being joined to a workgroup by the name of *DomainName*.
- The server MUST configure the local Windows Time Service (as specified in [WTSREF]) so that it is aware of being joined to a workgroup.
- If no errors occur, the server MUST return NERR_Success.

3.2.4.15 NetrUnjoinDomain2 (Opnum 23)

The **NetrUnjoinDomain2** method uses encrypted credentials to unjoin a computer from a workgroup or domain. This method replaces *NetrUnjoinDomain*.<101>

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

ServerName: Pointer to a Unicode string specifying the server. The server SHOULD<102> ignore this parameter. RPC client machines SHOULD <103> set this parameter to a string that resolves to an IP address of the server.

AccountName: 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: Pointer to a **JOINPR_ENCRYPTED_USER_PASSWORD** structure that specifies the encrypted password to use with the *AccountName* parameter. For information on decrypting and decoding the password, see section 2.2.4.18. This parameter is optional. If this parameter is NULL, the caller's security context MUST be used.

Options: A 32-bit bit field specifying modifications to default message processing behavior.

Value	Meaning
NETSETUP_ACCT_DELETE 0x00000004	Disables the account when the unjoin operation occurs.
NETSETUP_IGNORE_UNSUPPORTED_FLAGS 0x10000000	Windows Server 2003 and later: The server ignores undefined flags. This option is present to allow for the addition of new optional values in the future.

Return Values: The method MUST return 0x00000000 (NERR_Success) on success; otherwise, the following is a summary of the return values that an implementation MUST return, as specified by the following message processing. These error codes are specified in [MS-ERREF] section 3.

Return value/code	Description
0x00000005 ERROR_ACCESS_DENIED	Access is denied.
0x00000056 ERROR_INVALID_PASSWORD	The specified network password is not correct.
0x0000085C NERR_InternalError	An internal Windows error occurred.
0x00000A84 NERR_SetupNotJoined	This computer is not currently joined to a domain.
0x00000A85 NERR_SetupDomainController	This computer is a domain controller and cannot be unjoined from a domain.

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

The *computer-name* MUST be a locally persisted tuple where *computer-name.netbios* is a UTF-8 string of 15 characters or less. For more information, see section 3.2.1.1.

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. <104>

A computer is said to be joined to a domain if a certain state exists on the computer and in the domain NC. For more information about the specific state requirements that MUST occur both locally and in the domain NC at a domain controller, see [MS-ADTS] section 7.4.

The following ordered statements describe the sequence of message processing operations.

DomainControllerString: UTF-8 string containing the name of a domain controller in the domain that the server is joined.

DomainObject: An object in the domain database, as specified in [MS-ADTS] section 7.4, having the value of computer-account-name for the **SamAccountName** attribute.

PasswordString: A UTF-8 string containing a persisted password in cleartext.

The server SHOULD<105> ensure that the caller used the NCACN_NP RPC protocol sequence and SHOULD <106> return RPC_S_PROTSEQ_NOT_SUPPORTED otherwise. For information on how to identify the protocol sequence used, see [MS-RPCE].

The server SHOULD<107> enforce appropriate security measures to make sure that the caller's security context 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 SHOULD fail the call with ERROR_ACCESS_DENIED (0x00000005). For more information on how to determine the identity of the caller for the purpose of performing an access check, see 3.3.3.1.5.

If *Password* is NULL, then *PasswordString* MUST be NULL. Otherwise, the server MUST decrypt and decode the *Password*, as specified in section 2.2.4.18. *PasswordString* MUST equal the decrypted and decoded value. The decrypted buffer is represented as JOINPR_USER_PASSWORD, as specified in section 2.2.4.17. The Length field MUST be less than 513 bytes in length, otherwise message processing is stopped and the server MUST return ERROR_INVALID_PASSWORD.

The server MUST stop message processing and return NERR_SetupNotJoined if PrimaryDomain.Sid is NULL, as specified in [MS-LSAD].

If any bits other than NETSETUP_ACCT_DELETE are set in *Options*, the server checks for the bit NETSETUP_IGNORE_UNSUPPORTED_FLAGS. If not set, the server MUST stop message processing and return ERROR_INVALID_FLAGS. Otherwise, message processing continues.

The server MUST stop message processing and return NERR_SetupDomainController if the server processing the message is a domain controller. Otherwise, message processing continues.

The server MUST locate a domain controller in the joined domain<108>. *DomainControllerString* MUST be equal to the string name of the located domain controller.

The server establishes an authenticated SMB (as specified in [MS-RPCE]) session between the server and the domain controller named by the value of *DomainControllerString*. The security context established MUST be that of *AccountName*, and the credentials supplied during authentication are those specified in *PasswordString*. This security context is used by the domain controller for the purpose of performing access checks on all operations targeting the domain controller over this transport.

The server MUST configure the local Net Logon Remote Protocol (as specified in [MS-NRPC]) so that it is aware of no longer being joined to a domain.

The server MUST configure the local Windows Time Service (as specified in [WTSREF]) so that it is aware of being joined to a domain.

The server MUST set **PrimaryDomain.Sid** to NULL, as specified in [MS-LSAD].

The server MUST delete the persisted *PasswordString* that was stored previously when processing a **NetrJoinDomain2** message.

ComputerAccountString MUST equal the UTF-8 string consisting of the computer-name suffixed with a "$" character.

The server MUST update the *DomainObject* **userAccountControl** attribute by setting the bit USER_ACCOUNT_DISABLED to 1, as specified in [MS-SAMR].<109>

If the bit NETSETUP_ACCT_DELETE is set in *Options*, then the following LDAP attributes on domain-object MUST be set to the values in the table. The security context provided to the LDAP protocol is *AccountName* and the credential is *PasswordString*. For information about attributes and attribute names, see [MS-ADTS]. For information about LDAP, see [RFC2252] and [RFC2253].

LDAP Attribute Name	Value
userAccountControl	The following bits must be set to one. USER_ACCOUNT_DISABLED (0x00000001), as specified in [MS-SAMR]. See section 3.2.5.14.2 userAccountControl Mapping Table in [MS-SAMR] for information on the mapping of these bits in the LDAP protocol.

The server MUST configure the Certificate Auto Enrollment Service (as specified in [MSFT-AUTOENROLLMENT]) so that it is aware of not being joined to a domain.

The server MUST configure the local Windows Time Service (as specified in [WTSREF]) such that it is aware of not being joined to a domain.

The server MUST configure the local Net Logon Remote Protocol (as specified in [MS-NRPC]) such that it is aware of not being joined to a domain.

The server SHOULD store the value dns-name locally such that the Domain Naming System service unregisters name records for the local computer, as specified in [NIS].<110>

The server SHOULD remove the Domain Admins group to the local Administrators group, and the server SHOULD remove the Domain Users group to the local users groups, as specified in [MS-SAMR].<111>

If no errors occur, the server MUST return NERR_Success.

3.2.4.16 NetrRenameMachineInDomain2 (Opnum 24)

The **NetrRenameMachineInDomain2** method uses encrypted credentials to change the locally persisted *computer-name* and optionally rename a computer currently in a domain, without first removing the computer from the domain and then adding it back. This method replaces **NetrRenameMachineInDomain**.<112>

```
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: RPC binding handle (as specified in [C706]).

ServerName: Pointer to a Unicode string specifying the server. The server SHOULD<113> ignore this parameter. RPC client machines SHOULD <114> set this parameter to a string that resolves to an IP address of the server.

MachineName: 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: 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: Pointer to a JOINPR_ENCRYPTED_USER_PASSWORD structure that specifies the encrypted password to use with the *AccountName* parameter. For information on decrypting and decoding the password, see section 2.2.4.18. This parameter is optional. If this parameter is NULL, the caller's security context MUST be used.

Options: Specifies modifications to default behavior in message processing. The server MUST ignore any options that it does not understand.

Value	Meaning
NETSETUP_ACCT_CREATE 0x00000002	Renames the computer account in the domain. If this flag is not set, the computer account name is changed locally but no changes are made to the computer account object in the domain.

Return Values: The method MUST return 0x00000000 (NERR_Success) on success; otherwise, the following is a summary of the return values that an implementation MUST return as specified by the message processing below. These error codes are specified in [MS-ERREF] section 3.

Return value/code	Description
0x00000005 ERROR_ACCESS_DENIED	Access is denied.
0x00000008 ERROR_NOT_ENOUGH_MEMORY	Not enough storage is available to process this command.
0x00000032 ERROR_NOT_SUPPORTED	The request is not supported.
0x00000056 ERROR_INVALID_PASSWORD	The specified network password is not correct.
0x00000057 ERROR_INVALID_PARAMETER	The parameter is incorrect.
0x000008B0 NERR_UserExists	The account already exists.
0x00000A84 NERR_SetupNotJoined	This computer is not currently joined to a domain.
0x000006A7 RPC_S_PROTSEQ_NOT_SUPPORTED	The RPC protocol sequence is not supported.

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

The *computer-name* MUST be a locally persisted tuple where *computer-name.netbios* is a UTF-8 string of 15 characters or fewer. See section 3.2.1.1 for more information.

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. <115>

The following are ordered statements describing the sequence of message processing operations.

DomainNameString: UTF-8 string containing the name of the domain the server is joined to.

DomainControllerString: UTF-8 string containing the name of a domain controller in the domain that server is joined.

DomainObject: An object in the domain database as specified in [MS-ADTS] section 7.4.

OldComputerAccountString: UTF-8 string containing the value computer-name.netbios concatenated with a '$' character.

NewComputerAccountString: UTF-8 string containing the value to be stored in the samAccountName attribute of the computer object in the domain database.

ComputerNameString: UTF-8 string containing the value of new NetBIOS name of the computer.

DNSComputerNameString: UTF-8 string containing the new DNS name of the computer.

Spn1: UTF-8 string.

Spn2: UTF-8 string.

PasswordString: A UTF-8 string containing a password in cleartext.

The server SHOULD<116> ensure that the caller used the NCACN_NP RPC protocol sequence and SHOULD <117> return RPC_S_PROTSEQ_NOT_SUPPORTED otherwise. See [MS-RPCE] for information on how to identify the protocol sequence used.

The server SHOULD<118> enforce appropriate security measures to make sure that the caller's security context 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 SHOULD fail the call with ERROR_ACCESS_DENIED (0x00000005). For more information on how to determine the identity of the caller for the purpose of performing an access check, see [MS-RPCE] section 3.3.3.1.5.

If *Password* is NULL then *PasswordString* MUST be NULL. Otherwise, the server MUST decrypt and decode the *Password* as specified in section 2.2.4.18. *PasswordString* MUST be equal to the decrypted and decoded value. The decrypted buffer is represented as JOINPR_USER_PASSWORD, as specified in section 2.2.4.17. The Length field MUST be less than 513 bytes in length, otherwise message processing is stopped and the server MUST return ERROR_INVALID_PASSWORD.

If the computer 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.

If the computer is an **RODC**, then the server MUST fail the call with ERROR_NOT_SUPPORTED. Otherwise, message processing continues.

If the *Options* parameter does not contain NETSETUP_ACCT_CREATE, the server MUST apply the new name locally updating *computer-name* so that other protocols on the same computer as the server can operate using the new name. If this operation fails, the server MUST return an error and stop message processing. If the operation is successful, then the server MUST stop message processing and return successfully.

If the *Options* parameter contains NETSETUP_ACCT_CREATE, the server MUST continue message processing.

The server MUST convert<119> 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 "$" character. *OldComputerAccountString* MUST equal the concatenation of the old NetBIOS name of the machine with "$" character.

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.

The server MUST locate a writable domain controller for the domain the computer is joined to. The name of the domain is stored in **PrimaryDomain.DomainName**. This writable domain controller MUST contain a *DomainObject* with the LDAP attribute samAccountName equal to *OldComputerAccountString*.<120> For information about LDAP attributes and values, see [MS-ADTS]. *DomainControllerString* MUST equal the string name of the located writable domain controller. <121>This writable domain controller SHOULD not contain a *DomainObject* with the LDAP attribute samAccountName equal to *NewComputerAccountString*. <122>

DNSComputerNameString MUST be the concatenation of *ComputerNameString* and the DNS suffix on the computer.<123>

SPN1 MUST be the concatenation of "HOST/" with *DNSComputerNameString*.

SPN2 MUST be the concatenation of "HOST/" with *ComputerNameString*.

The server MUST update the writable domain controller named by the value of *DomainNameString* with the following LDAP attributes on the *DomainObject*. The security context provided to the LDAP protocol is *AccountName* and the credential is *PasswordString*. For more information about attributes and attribute names, see [MS-ADTS]. For more information about LDAP, see [RFC2252] and [RFC2253].

- samAccountName updated to equal *NewComputerAccountString*.<124>
- dnsHostName updated to equal *DNSComputerNameString*.<125>
- servicePrincipalName updated to contain *SPN1* and *SPN2*.<126>

If any of these updates fail, the server MUST fail the request and return the error from the writable domain controller.

The server SHOULD <127> update the writable domain controller named by the value of *DomainNameString* with the following LDAP attribute on the *DomainObject*:

- displayName updated to equal *NewComputerNameString*.<128>

The server SHOULD continue processing if this optional update fails.<129>

The server MUST apply the new name locally, updating *computer-name* so other protocols on the same computer as the server can act using the new name.

If all succeeds, the server MUST return NERR_Success.

3.2.4.17 NetrValidateName2 (Opnum 25)

The **NetrValidateName2** method verifies the validity of a computer, workgroup, or domain name. This method replaces the **NetrValidateName** method.<130>

```
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: RPC binding handle (as specified in [C706]).

ServerName: Pointer to a Unicode string specifying the server. The server SHOULD<131> ignore this parameter. RPC client machines SHOULD <132> set this parameter to a string that resolves to an IP address of the server.

NameToValidate: Pointer to a string that specifies the name to validate, according to its type.

AccountName: The server SHOULD<133> ignore this parameter.

Password: The server SHOULD<134> ignore this parameter.

NameType: Specifies the type of validation to perform, as specified in section 2.2.2.2.

Return Values: The method MUST return 0x00000000 (NERR_Success) on success; otherwise, the following summarizes the return values that an implementation MUST return as specified by the following message processing. These error codes are specified in [MS-ERREF] section 3.

Return value/code	Description
0x00000005 ERROR_ACCESS_DENIED	Access is denied.
0x00000034 ERROR_DUP_NAME	The connection was denied because a duplicate name exists on the network. Go to **System** in **Control Panel** to change the computer name and try again.
0x00000056 ERROR_INVALID_PASSWORD	The specified network password is incorrect.
0x00000057 ERROR_INVALID_PARAMETER	The parameter is incorrect.
0x0000007B ERROR_INVALID_NAME	The file name, directory name, or volume label syntax is incorrect.
0x000004BC ERROR_INVALID_DOMAINNAME	The format of the specified domain name is invalid.
0x0000054B ERROR_NO_SUCH_DOMAIN	The specified domain either does not exist or could not be contacted.
0x0000092F	This computer name is invalid.

Return value/code	Description
NERR_InvalidComputer	
0x00000A87 NERR_InvalidWorkgroupName	The specified workgroup name is invalid.
0x00002554 DNS_ERROR_NON_RFC_NAME	The DNS name does not comply with RFC specifications.
0x00002558 DNS_ERROR_INVALID_NAME_CHAR	The DNS name contains an invalid character.

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

The following ordered statements describe the sequence of message processing operations.

PasswordString: A UTF-8 string containing a password in cleartext.

The server SHOULD <135> ensure that the caller used the NCACN_NP RPC protocol sequence and return RPC_S_PROTSEQ_NOT_SUPPORTED otherwise. For information on how to identify the protocol sequence used, see [MS-RPCE].

The server SHOULD <136> ensure that the caller is local. For information on how to identify whether the caller is local, see [MS-RPCE].

The server SHOULD<137> enforce appropriate security measures to make sure that the caller's security context 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 SHOULD fail the call. For information on how to determine the identity of the caller for the purpose of performing an access check, see [MS-RPCE] section 3.3.3.1.5.

If *Password* is NULL then *PasswordString* MUST be NULL. Otherwise, the server MUST decrypt and decode the *Password* as specified in section 2.2.4.18. *PasswordString* MUST equal the decrypted and decoded value. The decrypted buffer is represented as JOINPR_USER_PASSWORD, as specified in section 2.2.4.17. The Length field MUST be less than 513 bytes in length, otherwise message processing is stopped and the server MUST return ERROR_INVALID_PASSWORD.

If *NameType* is *NetSetupUnknown* the server MUST stop message processing and return ERROR_INVALID_PARAMETER.

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.
- Error 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.
- Error NERR_InvalidComputer MUST be returned if the check fails.

NetSetupDomain

- The name MUST NOT consist entirely of the dot and space characters.
  - Error 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, as specified by [RFC1035]. Error DNS_ERROR_NON_RFC_NAME MUST be returned if this restriction is violated.

NetSetupDnsMachine

The validation is performed as specified in [RFC1035] 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 (an octet is an 8-bit value).
- Contain a label longer than 63 octets.
- Contain two or more consecutive dots.

- Begin with a dot.
  - Error ERROR_INVALID_NAME MUST be returned if any condition in this group is violated.
- Contain a space.
- Contain any of the following characters:

```
{ | } ~ [ \ ] ^ ' : ; < = > ? @ ! " # $ % ^ ` ( ) + / , *
```

- Error DNS_ERROR_INVALID_NAME_CHAR MUST be returned if any condition in this group is violated.

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. Error NERR_InvalidWorkgroupName MUST be returned if this condition is violated.
- The name MUST be valid for registration as a NetBIOS group name, as specified in [RFC1001].<138> 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, as specified in [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 ERROR_DUP_NAME MUST be returned if this condition is violated.<139>

NetSetupDomain

- The name MUST differ from the name of the **built-in domain**, "BUILTIN" as specified in [MS-LSAT] section 3.1.1.3, "Builtin Domain Principal View"; the comparison MUST be case-insensitive. Error NERR_InvalidComputer MUST be returned if this condition is violated.
- The name MUST be a name of an existing domain. Error ERROR_NO_SUCH_DOMAIN MUST be returned if this condition is not satisfied.<140>

NetSetupNonExistentDomain

- The name MUST differ from the name of the built-in domain, "BUILTIN" as specified in [MS-LSAT] section 3.1.1.3, "Builtin Domain Principal View"; the comparison MUST be case-insensitive. Error 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 ERROR_DUP_NAME MUST be returned if this condition is not satisfied.<141>

3.2.4.18 NetrGetJoinableOUs2 (Opnum 26)

The **NetrGetJoinableOUs2** method returns a list of organizational units (OUs) in which the user can create an object. This method replaces the **NetrGetJoinableOUs** method.<142>

```
unsigned long NetrGetJoinableOUs2(
  [in] handle_t RpcBindingHandle,
  [in, string, unique] wchar_t* ServerName,
  [in, string] wchar_t* DomainName,
  [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: RPC binding handle (as specified in [C706]).

ServerName: Pointer to a Unicode string specifying the server. The server SHOULD<143> ignore this parameter. RPC client machines SHOULD <144> set this parameter to a string that resolves to an IP address of the server.

DomainName: Pointer to a string that specifies the root domain under which the method will search for OUs. This parameter is also the domain of the account that the *AccountName* parameter is in.

AccountName: 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 is used.

Password: Pointer to a **JOINPR_ENCRYPTED_USER_PASSWORD** structure that specifies the encrypted password to use with the *AccountName* parameter. See section 2.2.4.18 for information on decrypting and decoding the password. This parameter is optional. If the *AccountName* parameter is NULL, the caller's security context MUST used and this parameter MUST be ignored.

OUCount: Pointer to the count of OUs that the method returned. The server MUST ignore this parameter on input.

OUs: Pointer to a pointer of size *OUCount* to a block of OUStrings that are the joinable OUs that the method returned.

Return Values: The method MUST return 0x00000000 (NERR_Success) on success; otherwise, the following summarizes the return values that an implementation MUST return as specified by the message processing below. These error codes are specified in [MS-ERREF] section 3.

Return value/code	Description
0x00000005 ERROR_ACCESS_DENIED	Access is denied.
0x00000056 ERROR_INVALID_PASSWORD	The specified network password is incorrect.
0x00000057 ERROR_INVALID_PARAMETER	The parameter is incorrect.
0x0000085E NERR_InvalidAPI	The requested API is not supported on domain controllers.
0x8001011C	The requested protocol method is not callable remotely.

Return value/code	Description
RPC_E_REMOTE_DISABLED	

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

The following ordered statements describe the sequence of message processing operations.

PasswordString: A UTF-8 string containing a password in cleartext.

The server SHOULD<145> ensure that the caller used the NCACN_NP RPC protocol sequence and returns RPC_S_PROTSEQ_NOT_SUPPORTED otherwise. For information on how to identify the protocol sequence used, see [MS-RPCE].

The server SHOULD<146> ensure that the caller is local. For information on how to identify that the caller is local, See [MS-RPCE].

The server SHOULD<147> enforce appropriate security measures to make sure that the caller's security context 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 SHOULD fail the call with ERROR_ACCESS_DENIED (0x00000005). For information on how to determine the identity of the caller for the purpose of performing an access check, see [MS-RPCE] section 3.3.3.1.5.

If *Password* is NULL then *PasswordString* MUST be NULL. Otherwise, the server MUST decrypt and decode the *Password*, as specified in section 2.2.4.18. *PasswordString* MUST equal the decrypted and decoded value. The decrypted buffer is represented as JOINPR_USER_PASSWORD, as specified in section 2.2.4.17. The Length field MUST be less than 513 bytes in length, otherwise message processing is stopped and the server MUST return ERROR_INVALID_PASSWORD.

The server SHOULD<148> enforce that this call fails on a domain controller. Otherwise, message processing continues.

The server MUST locate a domain controller in the domain represented by *DomainName*.<149>

The server MUST query the domain controller for a list of all **distinguished names** of directory objects in the directory partition represented by *DomainName* that satisfy all of the following:

- Have an LDAP attribute with attribute name **objectClass** that contains the value "organizationalUnit".
- Have an LDAP attribute with attribute name **AllowedChildClassesEffective** that contains the value "computer".
- Have queries that use the context of the user identified by *DomainName*, *AccountName*, and *Password*. If *AccountName* and *Password* are NULL, then the caller's context will be used.

Descriptions of distinguished names, directory objects, directory partitions, attributes, attribute values, and attribute names are as specified in [MS-ADTS].<150>

The server MUST write the distinguished names returned by the query to OUs so that each distinguished name is represented by a NULL-terminated string. The server MUST write the count of strings written to OUs to *OUCount*.

3.2.4.19 NetrAddAlternateComputerName (Opnum 27)

The **NetrAddAlternateComputerName** method adds an alternate name for a specified server.<151>

```
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: RPC binding handle (as specified in [C706]).

ServerName: Pointer to a Unicode string specifying the server. The server SHOULD<152> ignore this parameter. RPC client machines SHOULD <153> set this parameter to a string that resolves to an IP address of the server.

AlternateName: Pointer to a string that specifies the new alternate name to add. The name MUST be a valid DNS host name, as specified in [RFC1035].

DomainAccount: 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 is used. If this parameter is specified, the format MUST be one of the following:

<NetBIOSDomainName>\<UserName>,

<FullyQualifiedDNSDomainName>\<UserName>, or

<UserName>@<FullyQualifiedDNSDomainName>

EncryptedPassword: Pointer to a **JOINPR_ENCRYPTED_USER_PASSWORD** structure that specifies the encrypted password to use with the *DomainAccount* parameter. For information on decrypting and decoding the password, see section 2.2.4.18. This parameter is optional. If the *DomainAccount* parameter is NULL, the caller's security context MUST be used and this parameter MUST be ignored.

Reserved: A 32-bit bit-field reserved for specifying modifications to default message processing behavior. This value is reserved and SHOULD<154> be set to zero.

Value	Meaning
NET_IGNORE_UNSUPPORTED_FLAGS 0x00000001	The responder will ignore flags it does not implement. Intended for future versions of the protocol.

Return Values: The method MUST return 0x00000000 (NERR_Success) on success; otherwise, the following is a summary of the return values that an implementation MUST return, as specified by the message processing below. These error codes are specified in [MS-ERREF] section 3.

Return value/code	Description
0x00000005 ERROR_ACCESS_DENIED	Access is denied.
0x00000008 ERROR_NOT_ENOUGH_MEMORY	Not enough storage is available to process this command.
0x00000056 ERROR_INVALID_PASSWORD	The specified network password is not correct.
0x00000032 ERROR_NOT_SUPPORTED	This method is not supported by this server.
0x000003EC ERROR_INVALID_FLAGS	Reserved contains an invalid value.
0x000006A7 RPC_S_PROTSEQ_NOT_SUPPORTED	The RPC protocol sequence is not supported.
0x000006FF RPC_S_CALL_IN_PROGRESS	A remote procedure call is already in progress <155>.

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

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. <156>

The following ordered statements describe the sequence of message processing operations.

DomainObject: An object in the domain database as described in [MS-ADTS] section 7.4.

NewAlternateNames: MUST be a new tuple entry for alternate-computer-names as specified in section 3.2.1.1.

PasswordString: A UTF-8 string containing a password in cleartext.

The server SHOULD<157> ensure that the caller used the NCACN_NP RPC protocol sequence and return RPC_S_PROTSEQ_NOT_SUPPORTED otherwise. For information on how to identify the protocol sequence used, see [MS-RPCE] .

The server SHOULD<158> enforce appropriate security measures to make sure that the caller's security context has required permissions 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. For information on how to determine the identity of the caller for the purpose of performing an access check, see [MS-RPCE] section 3.3.3.1.5.

The server SHOULD<159> stop message processing and return ERROR_NOT_SUPPORTED if the server is a client **SKU**. Otherwise, message processing continues.

If the bit NET_IGNORE_UNSUPPORTED_FLAGS is set in *Reserved*, then any other bits MUST be ignored. If any other bit is set in *Reserved* and this bit is not set, the server MUST return ERROR_INVALID_FLAGS.

If *EncryptedPassword* is NULL, then *PasswordString* MUST be NULL. Otherwise, the server MUST decrypt and decode the *EncryptedPassword* as specified in section 2.2.4.18. *PasswordString* MUST

be equal to the decrypted and decoded value. The decrypted buffer is represented as JOINPR_USER_PASSWORD, as specified in section 2.2.4.17. The Length field MUST be less than 513 bytes in length, otherwise message processing is stopped and the server MUST return ERROR_INVALID_PASSWORD.

The server MUST validate *AlternateName*. The validation is performed as specified in [RFC1035] in the following order. Specifically, the name MUST NOT:

- Be longer than 255 octets (an octet is an 8-bit value).
- Contain a label longer than 63 octets.
- Contain two or more consecutive dots.
- Begin with a dot.

Error ERROR_INVALID_NAME MUST be returned if any condition in the above group is violated. Otherwise, message processing continues.

- Contain a space.
- Contain any of the following characters:

```
{ | } ~ [ \ ] ^ ' : ; < = > ? @ ! " # $ % ^ ` ( ) + / , *
```

Error DNS_ERROR_INVALID_NAME_CHAR MUST be returned if any condition in the above group is violated. Otherwise, message processing continues.

NewAlternateNames.dns MUST be equal to AlternateName.

NewAlternateNames.netbios MUST be equal to *NewAlternateNames*.dns converted to a NetBIOS name.<160>

NewAlternateNames MUST be appended to the list in alternate-computer-names persisted locally such that the set of NetBIOS and DNS names currently assigned to this computer can be resolved on the network, as specified in [RFC1001] and [NIS].

If the server is joined to a domain as specified in [MS-ADTS] section 7.4, the server MUST make the following update in the domain.

The server MUST locate a writable domain controller for the domain. <161>

The following LDAP attributes on *DomainObject* MUST be set to the values in the table.<162> The security context provided to the LDAP protocol is *DomainAccount* and the credential is *PasswordString*. For more information about attributes and attribute names, see [MS-ADTS]. For more information about LDAP, see [RFC2252] and [RFC2253].

LDAP Attribute Name	Value
msDS-AdditionalDnsHostName	*NewAlternateNames*.dns

If an error occurs, message processing is stopped and the error is returned to the requester. Otherwise, message processing continues.

If all succeeds, the server MUST return NERR_Success.**Note** The value of NERR_Success is 0x00000000.

3.2.4.20 NetrRemoveAlternateComputerName (Opnum 28)

The **NetrRemoveAlternateComputerName** method removes an alternate name for a specified server.<163>

```
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: RPC binding handle (as specified in [C706]).

ServerName: Pointer to a Unicode string specifying the server. The server SHOULD<164> ignore this parameter. RPC client machines SHOULD <165> set this parameter to a string that resolves to an IP address of the server.

AlternateName: Pointer to a string that specifies the alternate name to remove. The name MUST be a valid DNS host name, as specified in [RFC1035].

DomainAccount: 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 is used. If this parameter is specified, the format MUST be one of the following:

- <NetBIOSDomainName>\<UserName>;
- <FullyQualifiedDNSDomainName>\<UserName>; or
- <UserName>@<FullyQualifiedDNSDomainName>.

EncryptedPassword: Pointer to a **JOINPR_ENCRYPTED_USER_PASSWORD** structure that specifies the encrypted password to use with the *DomainAccount* parameter. For information on decrypting and decoding the password, see section 2.2.4.18. This parameter is optional. If the *DomainAccount* parameter is NULL, the caller's security context MUST be used and this parameter MUST be ignored.

Reserved: A 32-bit bit-field reserved for specifying modifications to default message processing behavior. This value is reserved and SHOULD<166> be set to zero.

Value	Meaning
NET_IGNORE_UNSUPPORTED_FLAGS 0x00000001	The responder will ignore flags it does not implement. Intended for future versions of the protocol.

Return Values: The method MUST return 0x00000000 (**NERR_Success**) on success; otherwise, the following summarizes the return values that an implementation MUST return as specified by the message processing below. These error codes are specified in [MS-ERREF] section 2.2.

Return value/code	Description
0x00000005 ERROR_ACCESS_DENIED	Access is denied.
0x00000008 ERROR_NOT_ENOUGH_MEMORY	Not enough storage is available to process this command.
0x00000056 ERROR_INVALID_PASSWORD	The specified network password is not correct.
0x00000032 ERROR_NOT_SUPPORTED	This method is not supported by this server.
0x000003EC ERROR_INVALID_FLAGS	Reserved contains an invalid value.
0x00000490 ERROR_NOT_FOUND	*AlternateName* was not found in the current list of alternate names.
0x000006A7 RPC_S_PROTSEQ_NOT_SUPPORTED	The RPC protocol sequence is not supported.
0x000006FF RPC_S_CALL_IN_PROGRESS	A remote procedure call is already in progress <167>.

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

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. <168>

The following ordered statements describe the sequence of message processing operations.

DomainObject: MUST be an object in the domain database, as specified in [MS-ADTS] section 7.4.

OldAlternateNames: MUST be a tuple entry for alternate-computer-names, as specified in section 3.2.1.1.

PasswordString: A UTF-8 string containing a password in cleartext.

The server SHOULD<169> ensure that the caller used the NCACN_NP RPC protocol sequence and return RPC_S_PROTSEQ_NOT_SUPPORTED otherwise. For information on how to identify the protocol sequence used, see [MS-RPCE].

The server SHOULD<170> enforce appropriate security measures to make sure that the caller's security context 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 SHOULD fail the call. For information on how to determine the identity of the caller for the purpose of performing an access check, see [MS-RPCE] section 3.3.3.1.5.

The server SHOULD<171> return ERROR_NOT_SUPPORTED if the server is a client SKU configuration.

If the bit NET_IGNORE_UNSUPPORTED_FLAGS is set in *Reserved*, then any other bits MUST be ignored. If any other bit is set in *Reserved* and this bit is not set, the server MUST return **ERROR_INVALID_FLAGS**.

If *EncryptedPassword* is NULL, then *PasswordString* MUST be NULL. Otherwise, the server MUST decrypt and decode the *EncryptedPassword*, as specified in section 2.2.4.18. *PasswordString* MUST equal the decrypted and decoded value. The decrypted buffer is represented as **JOINPR_USER_PASSWORD**, as specified in section 2.2.4.17. The Length field MUST be less than 513 bytes in length, otherwise message processing is stopped and the server MUST return **ERROR_INVALID_PASSWORD**.

The server MUST validate *AlternateName*. The validation is performed as specified in [RFC1035] in the following order. Specifically, the name MUST NOT:

- Be longer than 255 octets (an octet is an 8-bit value);
- Contain a label longer than 63 octets;
- Contain two or more consecutive dots; or
- Begin with a dot.

Error **ERROR_INVALID_NAME** MUST be returned if any condition in the above group is violated. Otherwise, message processing continues.

- Contain a space.
- Contain any of the following characters:

```
{ | } ~ [ \ ] ^ ' : ; < = > ? @ ! " # $ % ^ ` ( ) + / , *
```

Error DNS_**ERROR_INVALID_NAME_CHAR** MUST be returned if any condition in the above group is violated. Otherwise, message processing continues.

The server MUST locate the tuple *OldAlternateNames* in alternate-computer-names as specified in section 3.2.1.1, where *OldAlternateNames*.dns MUST equal *AlternateName* and *OldAlternateNames*.netbios MUST equal *AlternateName* converted<172> to a NetBIOS name. If tuple *OldAlternateNames* cannot be found the server MUST return **ERROR_NOT_FOUND**.

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 DNS names currently assigned to this computer can be resolved on the network, as specified in [RFC1001] and [NIS].

If the server is joined to a domain as specified in [MS-ADTS] section 7.4, then the server MUST make the following update in the domain.

The server MUST locate a writable domain controller for the domain. <173>

DomainObject MUST be an object in the domain database at DomainController, as specified in [MS-ADTS] section 7.4.

The following LDAP attributes on domain-object MUST be set to remove the values in the table.<174> The security context provided to the LDAP protocol is *DomainAccount* and the credential is *PasswordString*. For more information about attributes and attribute names, see [MS-ADTS]. For more information about LDAP, see [RFC2252] and [RFC2253].

LDAP Attribute Name	Value
msDS-AdditionalDnsHostName	OldAlternateNames.dns

If an error occurs, message processing is stopped and the error is returned to the requester. Otherwise, message processing continues.

If all succeeds, the server MUST return NERR_Success, which is the value 0x00000000.

3.2.4.21 NetrSetPrimaryComputerName (Opnum 29)

The **NetrSetPrimaryComputerName** method sets the primary computer name for a specified server. <175>

```
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: RPC binding handle (as specified in [C706]).

ServerName: Pointer to a Unicode string specifying the server. The server SHOULD<176> ignore this parameter. RPC client machines SHOULD <177> set this parameter to a string that resolves to an IP address of the server.

PrimaryName: Pointer to a string that specifies the primary computer name to set. The name MUST be a valid DNS host name, as specified in [RFC1035].

DomainAccount: 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 is used. This parameter is not used if the server is not joined to a domain.

NetBIOSDomainName>\<UserName>,

<FullyQualifiedDNSDomainName>\<UserName>, or

<UserName>@<FullyQualifiedDNSDomainName>

EncryptedPassword: Pointer to a **JOINPR_ENCRYPTED_USER_PASSWORD** structure that specifies the encrypted password to use with the *DomainAccount* parameter. For information on decrypting and decoding the password, see section 2.2.4.18. This parameter is optional. If the *DomainAccount* parameter is NULL, the caller's security context MUST be used and this parameter MUST be ignored.

Reserved: A 32-bit bit-field reserved for specifying modifications to default message processing behavior. This value is reserved and SHOULD<178> be set to zero.

Value	Meaning
NET_IGNORE_UNSUPPORTED_FLAGS 0x00000001	The responder will ignore flags it does not implement. Intended for future versions of the protocol.

Return Values: The method MUST return 0x00000000 (NERR_Success) on success; otherwise, the following summarizes the return values that an implementation MUST return, as specified by the message processing below. These error codes are specified in [MS-ERREF] section 3.

Return value/code	Description
0x00000005 ERROR_ACCESS_DENIED	Access is denied.
0x00000008 ERROR_NOT_ENOUGH_MEMORY	Not enough storage is available to process this command.
0x00000056 ERROR_INVALID_PASSWORD	The specified network password is incorrect.
0x00000057 ERROR_INVALID_PARAMETER	The parameter is incorrect.
0x00000032 ERROR_NOT_SUPPORTED	This method is not supported by this server.
0x000003EC ERROR_INVALID_FLAGS	Reserved contains an invalid value.
0x000006A7 RPC_S_PROTSEQ_NOT_SUPPORTED	The RPC protocol sequence is not supported.
0x000006FF RPC_S_CALL_IN_PROGRESS	A remote procedure call is already in progress <179>.

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

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. <180>

The following ordered statements describe the sequence of message processing operations.

DomainObject: An object in the domain database as specified in [MS-ADTS] section 7.4.

OldAlternateNames: MUST be a tuple entry for alternate-computer-names as specified in section 3.2.1.1.

NewAlternateNames: MUST be a new tuple entry for alternate-computer-names as specified in section 3.2.1.1.

NetBIOSNameString: A UTF-8 string containing the value of *PrimaryName* converted to a NetBIOS name.

I: Unsigned integer used for indexing alternate-computer-names.

ComputerAccountString: UTF-8 string containing the value to be stored in the **samAccountName** attribute of the computer object in the domain database.

PasswordString: A UTF-8 string containing a password in cleartext.

The server SHOULD<181> ensure that the caller used the NCACN_NP RPC protocol sequence and return RPC_S_PROTSEQ_NOT_SUPPORTED otherwise. For information on how to identify the protocol sequence used, see [MS-RPCE].

The server SHOULD<182> enforce appropriate security measures to make sure that the caller's security context 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 SHOULD fail the call. For information on how to determine the identity of the caller for the purpose of performing an access check, see [MS-RPCE] section 3.3.3.1.5.

The server SHOULD<183> return ERROR_NOT_SUPPORTED if the server is a client SKU configuration.

If the bit NET_IGNORE_UNSUPPORTED_FLAGS is set in *Reserved*, then any other bits MUST be ignored. If any other bit is set in *Reserved* and this bit is not set, the server MUST return ERROR_INVALID_FLAGS.

If *EncryptedPassword* is NULL, then *PasswordString* MUST be NULL. Otherwise, the server MUST decrypt and decode the *EncryptedPassword*, as specified in section 2.2.4.18. *PasswordString* MUST equal the decrypted and decoded value. The decrypted buffer is represented as JOINPR_USER_PASSWORD, as specified in section 2.2.4.17. The Length field MUST be less than 513 bytes in length, otherwise message processing is stopped and the server MUST return ERROR_INVALID_PASSWORD.

The server MUST validate *PrimaryName*. The validation is performed as specified in [RFC1035] in the following order. Specifically, the name MUST NOT:

- Be longer than 255 octets (an octet is an 8-bit value).
- Contain a label longer than 63 octets.
- Contain two or more consecutive dots.
- Begin with a dot.

Error ERROR_INVALID_NAME MUST be returned if any condition in the above group is violated.

- Contain a space.
- Contain any of the following characters:

```
{ | } ~ [ \ ] ^ ' : ; < = > ? @ ! " # $ % ^ ` ( ) + / , *
```

Error DNS_ERROR_INVALID_NAME_CHAR MUST be returned if any condition in the above group is violated.

The server MUST convert<184> the name in the *PrimaryName* parameter to a string NetBIOS name. This conversion MUST match with the conversion used in [MS-NRPC]. *NetBIOSNameString* MUST be equal to this converted value.

The server MUST locate the tuple in the list of alternate-computer-names as specified in section 3.2.1.1, where alternate-computer-names [*I*].dns is equal to *PrimaryName* and alternate-computer-names[*I*].netbios MUST be equal to *NetBIOSNameString*. *OldAlternateNames* MUST be equal to the tuple identified.

The server MUST remove the tuple located above from the list in alternate-computer-names persisted locally.

NewAlternateNames.dns MUST be equal to the current computer-name.dns, as specified in section 3.2.1.1.

NewAlternateNames.netbios MUST be equal to the current computer-name.netbios, as specified in section 3.2.1.1.

NewAlternateNames MUST be appended to the list in alternate-computer-names persisted locally.

The server MUST set computer-name.netbios to equal *NetBIOSNameString*.

The server MUST set computer-name.dns to equal *PrimaryName*.

ComputerAccountString MUST be a UTF-8 string consisting of computer-name.netbios suffixed with a "$" character.

The server MUST store the values computer-name.netbios, computer-name.dns, and names in alternate-computer-names locally so that the set of NetBIOS and DNS names currently assigned to this computer can be resolved on the network, as specified in [RFC1001] and [NIS].

If the server is joined to a domain, as specified in [MS-ADTS] section 7.4, then the server MUST make the following updates in the domain.

The server MUST locate a writable domain controller for the domain.<185>

The following LDAP attributes on *DomainObject* MUST be set to the values in the table.<186> The security context provided to the LDAP protocol is *DomainAccount* and the credential is *PasswordString*. For more information about attributes and attribute names, see [MS-ADTS]. For more information about LDAP, see [RFC2252] and [RFC2253].

LDAP attribute name	Value
SamAccountName<187>	*ComputerAccountString*
dNSHostName	computer-name.dns
msDS-AdditionalDnsHostName	Remove *OldAlternateNames*.dns and add *NewAlternateNames*.dns

If an error occurs, message processing stops and the error is returned to the requester. Otherwise, message processing continues.

If all succeeds, the server MUST return NERR_Success.

Note the value of NERR_Success is 0x00000000.

3.2.4.22 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.<188>

```
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: Pointer to a Unicode string specifying the server. The client MUST map this string to an RPC binding handle, and the server MUST ignore this argument. For more information, see [C706] sections 4.3.5 and 5.1.5.2.

NameType: The type of query issued. For more information, see **NET_COMPUTER_NAME_TYPE (section 2.2.2.3)**.

Reserved: A 32-bit bit-field reserved for specifying modifications to default message processing behavior. This value is reserved and SHOULD<189> be set to zero.

Value	Meaning
NET_IGNORE_UNSUPPORTED_FLAGS 0x00000001	The responder will ignore flags it does not implement. Intended for future versions of the protocol.

ComputerNames: Pointer to structure containing a list of computer name strings. For more information, see section 2.2.4.20.

Return Values: The method MUST return 0x00000000 (NERR_Success) on success; otherwise, the following summarizes the return values that an implementation MUST return as specified by the message processing below. These error codes are specified in [MS-ERREF] section 3.

Return value/code	Description
0x00000005 ERROR_ACCESS_DENIED	Access is denied.
0x00000008 ERROR_NOT_ENOUGH_MEMORY	Not enough storage is available to process this command.
0x00000057 ERROR_INVALID_PARAMETER	The parameter is incorrect.
0x00000032 ERROR_NOT_SUPPORTED	This method is not supported by this server.
0x000003EC ERROR_INVALID_FLAGS	Reserved contains an invalid value.
0x000006A7 RPC_S_PROTSEQ_NOT_SUPPORTED	The RPC protocol sequence is not supported.

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

The following ordered statements describe the sequence of message processing operations.

The server SHOULD<190> ensure that the caller used the NCACN_NP RPC protocol sequence and return RPC_S_PROTSEQ_NOT_SUPPORTED otherwise. For information on how to identify the protocol sequence used, see [MS-RPCE].

The server SHOULD<191> enforce appropriate security measures to make sure that the caller's security context 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 SHOULD fail the call. For information on how to determine the identity of the caller for the purpose of performing an access check, see [MS-RPCE] section 3.3.3.1.5.

The server SHOULD<192> return ERROR_NOT_SUPPORTED if the server is a client SKU configuration.

If the bit NET_IGNORE_UNSUPPORTED_FLAGS is set in *Reserved*, then any other bits MUST be ignored. If any other bit is set in *Reserved* and this bit is not set, the server MUST return ERROR_INVALID_FLAGS.

The server MUST return ERROR_INVALID_PARAMETER if NameType is greater than or equal to NetComputerNameTypeMax.

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 equal to 1 and initialize the UNICODE_STRING values in *ComputerNames* to equal computer-name.dns.
- NetAlternateComputerNames:
  - The server MUST set ComputerNames.EntryCount equal to the number of tuples contained in alternate-computer-names. 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].dns.
- NetAllComputerNames:
  - The server MUST set ComputerNames.EntryCount equal to the number of tuples contained in alternate-computer-names + 1. The ComputerNames array must be initialized to return all the names as specified in NetPrimaryComputerName and NetAlternateComputerNames above.

If all succeeds, the server MUST return NERR_Success.

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

There are no local events used on the server beyond the events maintained in the underlying RPC transport.

4 Protocol Examples

4.1 NetrWkstaGetInfo Example

As an example, the client calls the **NetrWkstaGetInfo** method on a server named "srvr1.example.com":

```
NetrWkstaGetInfo (
    [in,string,unique] WKSSVC_IDENTIFY_HANDLE ServerName
            = "some.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 = ERROR_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 = "some.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** 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** 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 will 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 and passes a non-NULL pointer as *TotalEntries*. The client also passes the unchanged value of *ResumeHandle* (0x000000120).

On receiving this method, the server executes the method locally to continue enumeration based on a *ResumeHandle* value of 0x00000120, and returns ERROR_SUCCESS. The server returns the names of the next three 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 name "DomainA" and a domain controller of that domain named "DC-A".

A client calls the **NetrJoinDomain2** 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 SID properties of "DC-A", and stores them locally.

As a last step, "SrvrA" creates a domain-object as specified in [MS-ADTS] section 7.4.2 for itself, and sets the following attributes:

- sAMAccountName
- userAccountControl
- unicodePwd
- dNSHostName
- servicePrincipalName

The server then responds back 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 may be exchanged as a result of a **NetrJoinDomain2** message sent to the server.

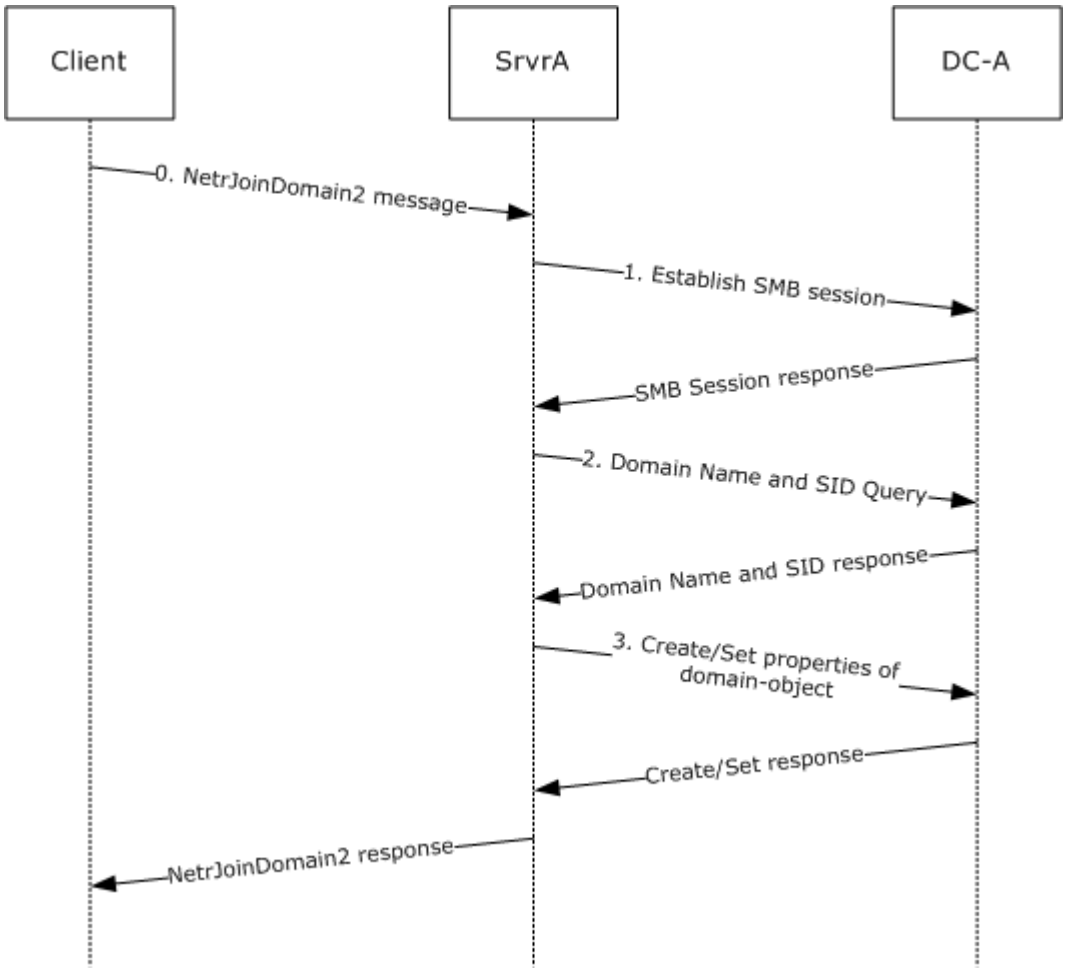

Figure 4: NetrJoinDomain2 sequence

The following state changes will occur as a part of this message:

- Initial states:

SrvrA	DC-A
PrimaryDomain.DomainName = "Workgroup". PrimaryDomain.SID = NULL. Domain-Secret-Value = NULL.	PrimaryDomain.DomainName = "DomainA". PrimaryDomain.SID = { SID-Value}.

- End states:

SrvrA	DC-A

PrimaryDomain.DomainName = "DomainA".
PrimaryDomain.SID = {SID-Value}.
Domain-Secret-Value = { domain-secret }.
	PrimaryDomain.DomainName = "DomainA".
	PrimaryDomain.SID = { SID-Value }.
	A computer account object exists with the following LDAP attributes:
	▪ 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

The following sections specify security considerations for implementers of the Workstation Service Remote Protocol.

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. The server implementation should enforce security on each method.

5.2 Index of Security Parameters

Security Parameter	Section
Authentication Protocol	2.1

5.3 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)**. If a PRNG is used, it should be compliant with recognized standards, such as FIPS 140-2 Annex C, as specified in [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;
```

```
    int 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;

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
);

void Opnum8NotUsedOnWire(void);

void Opnum9NotUsedOnWire(void);

void Opnum10NotUsedOnWire(void);

void Opnum11NotUsedOnWire(void);

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);

unsigned long
NetrJoinDomain(
    [in,string,unique] WKSSVC_IMPERSONATE_HANDLE ServerName,
    [in,string] wchar_t* DomainName,
    [in,string,unique] wchar_t* MachineAccountOU,
    [in,string,unique] wchar_t* AccountName,
    [in,string,unique] wchar_t* Password,
    [in] unsigned long Options
);

unsigned long
NetrUnjoinDomain(
    [in,string,unique] WKSSVC_IMPERSONATE_HANDLE ServerName,
    [in,string,unique] wchar_t* AccountName,
    [in,string,unique] wchar_t* Password,
    [in] unsigned long Options
);

unsigned long
NetrRenameMachineInDomain(
    [in,string,unique] WKSSVC_IMPERSONATE_HANDLE ServerName,
    [in,string,unique] wchar_t* MachineName,
    [in,string,unique] wchar_t* AccountName,
    [in,string,unique] wchar_t* Password,
    [in] unsigned long Options
);
```

```
    unsigned long
    NetrValidateName(
        [in,string,unique] WKSSVC_IMPERSONATE_HANDLE ServerName,
        [in,string] wchar_t* NameToValidate,
        [in,string,unique] wchar_t* AccountName,
        [in,string,unique] wchar_t* Password,
        [in] NETSETUP_NAME_TYPE NameType
    );

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

    unsigned long
    NetrGetJoinableOUs(
        [in,string,unique] WKSSVC_IMPERSONATE_HANDLE ServerName,
        [in,string] wchar_t*    DomainName,
        [in,string,unique] wchar_t*    AccountName,
        [in,string,unique] wchar_t*    Password,
        [in,out] unsigned long*    OUCount,
        [out,string,size_is(,*OUCount)] wchar_t**    OUs
    );


#define JOIN_OBFUSCATOR_LENGTH 8
#define JOIN_MAX_PASSWORD_LENGTH 256

    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;

    unsigned long
    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
    );

    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*  DomainName,
    [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
    );

    void Opnum31NotUsedOnWire(void);

}
```

7 Appendix B: Windows Behavior

The information in this specification is applicable to the following versions of Windows:

- Windows NT
- Windows 2000
- Windows XP
- Windows Server 2003
- Windows Vista

Exceptions, if any, are noted below. Unless otherwise specified, any statement of optional behavior in this specification prescribed using the terms SHOULD or SHOULD NOT implies Windows behavior in accordance with the SHOULD or SHOULD NOT prescription. Unless otherwise specified, the term MAY implies that Windows does not follow the prescription.

<1> Section 1.4: Windows implementations use [MS-SAMR] to manipulate the domain computer account object.

<2> Section 1.4: Windows implementations use the domain locator protocol, as specified in [MS-ADTS] section 7.3.6, when communication to a domain controller is specified.

<3> Section 1.8: Windows only uses the values in [MS-ERREF].

<4> Section 2.1: Windows uses the identity of the caller to perform method-specific access checks.

<5> Section 2.2.4.8: The case-sensitive name of the NetBIOS transport protocol in the Windows implementation is "NetBIOS".

<6> Section 2.2.4.8: Windows NT, Windows 2000, and Windows Server 2003 implementations set wkti0_transport_address for NetBT transport protocol to a string that represents the IEEE 802.1 Media Access Control (MAC) address of the transport protocol.

<7> Section 2.2.4.9: In the Windows implementation, the name of the user is the account name that was used to authenticate the user on the computer.

<8> Section 2.2.4.11: On Windows 2000, Windows XP, Windows Server 2003, and Windows Vista, **RawReadsDenied** may contain any value on send and is ignored on receipt.

<9> Section 2.2.4.11: On Windows 2000, Windows XP, Windows Server 2003, and Windows Vista, **RawWritesDenied** may contain any value on send and is ignored on receipt.

<10> Section 3.2.1.1: Windows XP, Windows Server 2003, and Windows Vista implement alternate-computer-names. The default state for this list is empty.

<11> Section 3.2.4: The server uses the underlying Windows security subsystem to determine the permissions for the caller.

<12> 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.

Opnum	Description
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.
31	Only used locally by Windows, never remotely.

<13> Section 3.2.4.1: Windows requires that if the *Level* parameter is equal to 0x00000066 or 0x000001F6, then the caller must be a member of the Administrators group. If the caller is not a member of the Administrators group, then the server fails the method with ERROR_ACCESS_DENIED (0x00000005).

<14> Section 3.2.4.2: For these *Level* values Windows non-deterministically returns either STATUS_SUCCESS or STATUS_INVALID_PARAMETER.

<15> Section 3.2.4.2: Windows never uses this value to configure the SMB redirector. The server stores the value and returns it when the client requests it.

<16> Section 3.2.4.2: Windows never uses this value to configure the SMB redirector. The server stores the value and returns it when the client requests it.

<17> Section 3.2.4.2: Windows never uses this value to configure the SMB redirector. The server stores the value and returns it when the client requests it.

<18> Section 3.2.4.2: Windows never uses this value to configure the SMB redirector. The server stores the value and returns it when the client requests it.

<19> Section 3.2.4.2: Windows never uses this value to configure the SMB redirector. The server stores the value and returns it when the client requests it.

<20> Section 3.2.4.2: Windows never uses this value to configure the SMB redirector. The server stores the value and returns it when the client requests it.

<21> Section 3.2.4.2: Windows never uses this value to configure the SMB redirector. The server stores the value and returns it when the client requests it.

<22> Section 3.2.4.2: Windows never uses this value to configure the SMB redirector. The server stores the value and returns it when the client requests it.

<23> Section 3.2.4.2: Windows never uses this value to configure the SMB redirector. The server stores the value and returns it when the client requests it.

<24> Section 3.2.4.2: Windows never uses this value to configure the SMB redirector. The server stores the value and returns it when the client requests it.

<25> Section 3.2.4.2: Windows never uses this value to configure the SMB redirector. The server stores the value and returns it when the client requests it.

<26> Section 3.2.4.2: Windows never uses this value to configure the SMB redirector. The server stores the value and returns it when the client requests it.

<27> Section 3.2.4.2: Windows never uses this value to configure the SMB redirector. The server stores the value and returns it when the client requests it.

<28> Section 3.2.4.2: Windows never uses this value to configure the SMB redirector. The server stores the value and returns it when the client requests it.

<29> Section 3.2.4.2: Windows never uses this value to configure the SMB redirector. The server stores the value and returns it when the client requests it.

<30> Section 3.2.4.2: Windows never uses this value to configure the SMB redirector. The server stores the value and returns it when the client requests it.

<31> Section 3.2.4.2: Windows never uses this value to configure the SMB redirector. The server stores the value and returns it when the client requests it.

<32> Section 3.2.4.2: Windows never uses this value to configure the SMB redirector. The server stores the value and returns it when the client requests it.

<33> Section 3.2.4.2: Windows never uses this value to configure the SMB redirector. The server stores the value and returns it when the client requests it.

<34> Section 3.2.4.2: Windows never uses this value to configure the SMB redirector. The server stores the value and returns it when the client requests it.

<35> Section 3.2.4.2: Windows never uses this value to configure the SMB redirector. The server stores the value and returns it when the client requests it.

<36> Section 3.2.4.2: Windows never uses this value to configure the SMB redirector. The server stores the value and returns it when the client requests it.

<37> Section 3.2.4.2: Windows never uses this value to configure the SMB redirector. The server stores the value and returns it when the client requests it.

<38> Section 3.2.4.2: Windows never uses this value to configure the SMB redirector. The server stores the value and returns it when the client requests it.

<39> Section 3.2.4.2: Windows never uses this value to configure the SMB redirector. The server stores the value and returns it when the client requests it.

<40> Section 3.2.4.2: Windows never uses this value to configure the SMB redirector. The server stores the value and returns it when the client requests it.

<41> Section 3.2.4.2: Windows never uses this value to configure the SMB redirector. The server stores the value and returns it when the client requests it.

<42> Section 3.2.4.2: Windows never uses this value to configure the SMB redirector. The server stores the value and returns it when the client requests it.

<43> Section 3.2.4.2: Windows never uses this value to configure the SMB redirector. The server stores the value and returns it when the client requests it.

<44> Section 3.2.4.2: Windows never uses this value to configure the SMB redirector. The server stores the value and returns it when the client requests it.

<45> Section 3.2.4.2: Windows requires that the caller MUST be a member of the Administrators group. If the caller is not a member of the Administrators group then the server fails the method with ERROR_ACCESS_DENIED.

<46> Section 3.2.4.3: The Windows implementation returns the zero-based index of the user to be enumerated from the list.

<47> Section 3.2.4.3: Windows requires that the caller must be a member of the Administrators group.

<48> Section 3.2.4.3: The server identifies each 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** is called with a non-zero *ResumeHandle*, then the server only enumerates those active users who have a logon number greater than the *ResumeHandle* .

<49> Section 3.2.4.4: The Windows implementation returns the zero-based index of the transport protocol to be enumerated from the list.

<50> Section 3.2.4.4: The server maintains 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** 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** calls, then some transports may not be enumerated at all, or some transports may be enumerated multiple times.

<51> Section 3.2.4.5: On Windows 2000, Windows XP, Windows Server 2003, and Windows Vista, this method is deprecated. If the *Level* parameter is set to 0x00000000 and the caller belongs to the administrator group, then these servers always return ERROR_SUCCESS (0x00000000) without any further processing.

<52> Section 3.2.4.5: Windows NT 4.0 requires that the caller must be a member of the Administrators group.

<53> Section 3.2.4.6: On Windows 2000, Windows XP, Windows Server 2003, and Windows Vista, this method is deprecated. If the *ForceLevel* parameter is set to equal 0x00000000, 0x00000001, or 0x00000002, and the caller belongs to the Administrators group, then these servers always return ERROR_SUCCESS (0x00000000).

<54> Section 3.2.4.6: Windows NT 4.0 requires that the caller must be a member of the Administrators group.

<55> Section 3.2.4.7: Windows implements these implementation-specific values as counters of 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.

<56> Section 3.2.4.8: Windows implementations do not support this method. All calls to this method return ERROR_NOT_SUPPORTED (0x00000032).

<57> Section 3.2.4.8: This method was added in Windows 2000 and is in all subsequent versions.

<58> Section 3.2.4.8: Windows implementations do not support this method. All calls to this method return ERROR_NOT_SUPPORTED (0x00000032).

<59> Section 3.2.4.9: Windows implementations do not support this method. All calls to this method return ERROR_NOT_SUPPORTED (0x00000032).

<60> Section 3.2.4.9: This method was added in Windows 2000 and is in all subsequent versions.

<61> Section 3.2.4.9: Windows implementations do not support this method. All calls to this method return ERROR_NOT_SUPPORTED (0x00000032).

<62> Section 3.2.4.10: Windows implementations do not support this method. All calls to this method return ERROR_NOT_SUPPORTED (0x00000032).

<63> Section 3.2.4.10: This method was added in Windows 2000 and is in all subsequent versions.

<64> Section 3.2.4.10: Windows implementations do not support this method. All calls to this method return ERROR_NOT_SUPPORTED (0x00000032).

<65> Section 3.2.4.11: Windows implementations do not support this method. All calls to this method return ERROR_NOT_SUPPORTED (0x00000032).

<66> Section 3.2.4.11: This method was added in Windows 2000 and is in all subsequent versions.

<67> Section 3.2.4.11: Windows implementations do not support this method. All calls to this method return ERROR_NOT_SUPPORTED (0x00000032).

<68> Section 3.2.4.12: Windows implementations ignore this parameter.

<69> Section 3.2.4.12: The verification of the proper RPC protocol sequence is enforced only on Windows XP SP2, Windows Server 2003 SP1, and Windows Vista.

<70> Section 3.2.4.12: The verification of the proper RPC protocol sequence is enforced only on Windows XP SP2, Windows Server 2003 SP1, and Windows Vista.

<71> Section 3.2.4.12: Windows requires that the caller be an authenticated user, and returns ERROR_ACCESS_DENIED if failed.

<72> Section 3.2.4.13: Windows implementations do not support this method. All calls to this method return ERROR_NOT_SUPPORTED (0x00000032).

<73> Section 3.2.4.13: This method was added in Windows 2000 and is in all subsequent versions.

<74> Section 3.2.4.13: Windows implementations do not support this method. All calls to this method return ERROR_NOT_SUPPORTED (0x00000032).

<75> Section 3.2.4.14: This method was added in Windows 2000 and is available in all subsequent versions.

<76> Section 3.2.4.14: Windows implementations ignore this parameter.

<77> Section 3.2.4.14: Windows RPC client machines set the *ServerName* parameter to a string that resolves to the IP protocol layer destination address of the RPC packets it transmits. The string formats are as follows:

1. An IP address (e.g. "10.0.0.23")

2. The fully qualified DNS name of the server

3. The NetBIOS name of the server

This RPC client behavior applies to all Windows RPC client machines, which includes Windows XP, Windows 2000, Windows 2000 Server, Windows Server 2003, Windows Vista, and Windows Server 2008.

<78> Section 3.2.4.14: Windows XP, Windows Server 2003, Windows Vista, and Windows Server 2008 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 NetrRenameMachineInDomain.

<79> Section 3.2.4.14: Windows XP, Windows Server 2003, Windows Vista, and Windows Server 2008 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 NetrRenameMachineInDomain.

<80> Section 3.2.4.14: Windows Vista and Windows Server 2008 implementations will 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.

<81> Section 3.2.4.14: Windows implementations support NETSETUP_JOIN_DC_ACCOUNT beginning with Windows Vista. Clients running Windows Server 2008 pass NETSETUP_JOIN_DC_ACCOUNT, which servers prior to Windows Vista ignore. Windows Server 2008 clients, when setting this flag, always locate a domain controller of Windows Server 2008 version or higher. The location mechanism is specified in [MS-ADTS] Section 7.3 "Publishing and Locating a Domain Controller", and the flag description for DS_FULL_SECRET_DOMAIN_6_FLAG, as specified in [MS-ADTS] section 7.3.3.2. The use of DS_FULL_SECRET_DOMAIN_6_FLAG ensures location of a domain controller with a minimum version of Windows Server 2008.

<82> Section 3.2.4.14: Windows Vista and Windows Server 2008 implementations always use the most recently set computer name during a domain join when NETSETUP_JOIN_WITH_NEW_NAME is specified. In the Windows implementation of a computer rename, a computer must be 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 3.2.4.16)** message would change the persisted abstract state **computer-name** (see section 3.2.1.1), but the change would not be effective until after a machine restart. Specifying this flag would cause the join operation to use the new **computer-name** when joining the domain.

<83> Section 3.2.4.14: Windows Vista and later: Indicates that the method SHOULD prevent concurrent calls

<84> Section 3.2.4.14.1: Windows implementations save the original state in memory for the duration of message processing before making any changes. 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 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 immediately before the call was processed.

<85> Section 3.2.4.14.1: Clients running Windows XP, Windows Server 2003, Windows Vista, and Windows Server 2008 pass **NETSETUP_MACHINE_PWD_PASSED** or **NETSETUP_DEFER_SPN**, which is ignored by servers earlier than Windows XP.

<86> Section 3.2.4.14.1: Windows clients of version Windows Server 2003, Windows Vista, and Windows Server 2008 define NETSETUP_IGNORE_UNSUPPORTED_FLAGS but do not set the flag. This flag is ignored by Windows servers earlier than Windows Server 2003.

<87> Section 3.2.4.14.1: Windows implementations support **NETSETUP_JOIN_DC_ACCOUNT** beginning with Windows Vista. Clients running Windows Server 2008 pass NETSETUP_JOIN_DC_ACCOUNT, which is ignored by servers earlier than Windows Vista. Windows Server 2008 clients, when setting this flag, always locate a domain controller of Windows Server 2008 version or higher. The location mechanism is specified in [MS-ADTS] section 7.3 and the flag description for **DS_FULL_SECRET_DOMAIN_6_FLAG**, as specified in [MS-ADTS] section 7.3.3.2. The use of DS_FULL_SECRET_DOMAIN_6_FLAG ensures location of a domain controller with a minimum version of Windows Server 2008.

<88> Section 3.2.4.14.1: Only on Windows XP SP2, Windows Server 2003 SP1, and Windows Vista, is the verification of the proper RPC protocol sequence enforced.

<89> Section 3.2.4.14.1: Only on Windows XP SP2, Windows Server 2003 SP1, and Windows Vista, is the verification of the proper RPC protocol sequence enforced.

<90> Section 3.2.4.14.1: Only on Windows XP SP2, Windows Server 2003 SP1, and Windows Vista, does Windows require that the caller be a member of the administrator group.

<91> Section 3.2.4.14.1: Windows Vista Home Basic and Windows Vista Home Premium return ERROR_NOT_SUPPORTED if this method is invoked.

<92> Section 3.2.4.14.2: Windows implementations find and set the described state through the following sequence:

The PrimaryDomain.DomainName and PrimaryDomain.Sid are accessible via the [MS-LSAD] protocol methods LsarQueryInformationPolicy and LsarQueryInformationPolicy2 by specifying PolicyDnsDomainInformation and PolicyPrimaryDomainInformation for the InformationClass parameter and targeting a domain controller. Likewise, these values are persisted using the LsaSetInformationPolicy and LsaSetInformationPolicy2 methods using the same values for InformationClass and targeting the machine joining the domain.

The *domain-secret* is persisted using the LsaOpenSecret and LsaSetSecret methods of the [MS-LSAD] protocol.

The computer account object is created at a writable domain controller using the SamrCreateUser2InDomain method of the [MS-SAMR] protocol.

The LDAP attributes userAccountControl and unicodePwd, as specified in [MS-ADA3] sections 1.340 and 1.330, respectively, are set using the SamrSetInformationUser2 method of the [MS-SAMR] protocol.

The LDAP attributes dnsHostName and servicePrincipalName, as specified in [MS-ADTS] section 1.63 and [MS-ADA3] section 1.252, respectively. They are set using the LDAP protocol (for more information, see [RFC2252] and [RFC2253]).

<93> Section 3.2.4.14.3: Windows implementations require a writable domain controller. For descriptions of a writable domain controller, see [MS-ADTS]. Windows implementations use the domain controller locator service specified in [MS-ADTS] to locate a writable domain controller.

<94> Section 3.2.4.14.3: Windows implementations use the domain controller locator service described in [MS-ADTS] to request that the named domain controller perform the location when the caller has specified a domain controller. Windows implementations first request the DNS name of the domain controller. If the name in the response does not match the name specified by the caller, a second request is sent requesting the NetBIOS name of the domain controller. If the name returned and DomainControllerString are not equal, the server SHOULD stop message processing and return **ERROR_INVALID_DOMAIN_ROLE**. Otherwise, message processing continues.

<95> Section 3.2.4.14.3: Windows implementations send a **LsarOpenPolicy2** request to a domain controller, and using the returned policy handle (see [MS-LSAD]) query for PrimaryDomain.DomainName and PrimaryDomain.Sid by sending a **LsarQueryInformationPolicy2** request for **InformationClass PolicyDnsDomainInformation**, followed by sending a **LsarQueryInformationPolicy** request for InformationClass **PolicyPrimaryDomainInformation**.

<96> Section 3.2.4.14.3: 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 that can use either DES or SHA1. Windows uses a SHA1-based PRNG to satisfy FIPS 140-2 level 2 cryptographic module certification requirements. In the PRNG description of Appendix 3.1, G is constructed from SHA1 with the first parameter as the initial value for the SHA1 registers, and 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").

XSEEDj 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 XSEEDj. The Windows implementation sets it to the shortest possible value, 1.

<97> Section 3.2.4.14.3: Windows implementations set **USER_ACCOUNT_DISABLED** bit to be zero using the Security Account Manager Remote Protocol (as specified in [MS-SAMR]).

<98> Section 3.2.4.14.3: Windows implementations use the LDAP protocol (as specified in [RFC1777]) to modify the LDAP attributes named dNSHostName and servicePrincipalName.

<99> Section 3.2.4.14.3: Windows implementations store the fully qualified DNS name of the computer locally based on a local configurable setting, which, by default, is set to store the name.

<100> Section 3.2.4.14.3: Windows implementations add the Domain Admins group to the local Administrators group and add the Domain Users group to the local users groups, as specified in [MS-SAMR].

<101> Section 3.2.4.15: This method was added in Windows 2000 and is available in all subsequent versions.

<102> Section 3.2.4.15: Windows implementations ignore this parameter.

<103> Section 3.2.4.15: Windows RPC client machines set the *ServerName* parameter to a string that resolves to the IP protocol layer destination address of the RPC packets it transmits. The string formats are as follows:

1. An IP address (for example, "10.0.0.23")
2. The fully qualified DNS name of the server
3. The NetBIOS name of the server

This RPC client behavior applies to all Windows RPC client machines, which includes Windows XP, Windows 2000, Windows 2000 Server, Windows Server 2003, Windows Vista, and Windows Server 2008.

<104> Section 3.2.4.15: 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

<105> Section 3.2.4.15: Only on Windows XP SP2, Windows Server 2003 SP1, and Windows Vista, is the verification of the proper RPC protocol sequence enforced.

<106> Section 3.2.4.15: Only on Windows XP SP2, Windows Server 2003 SP1, and Windows Vista, is the verification of the proper RPC protocol sequence enforced.

<107> Section 3.2.4.15: Windows requires that the caller be a member of the Administrators group.

<108> Section 3.2.4.15: Windows implementations require a writable domain controller. For descriptions of a writable domain controller, see [MS-ADTS]. Windows implementation uses the Domain Controller Locator service specified in [MS-ADTS] to locate a writable domain controller.

<109> Section 3.2.4.15: Windows implementations set USER_ACCOUNT_DISABLED bit to 1 using the Security Account Manager (SAM) Remote Protocol (as specified in [MS-SAMR]).

<110> Section 3.2.4.15: Windows implementations update the fully qualified DNS name of the computer locally based on a local configurable setting, which, by default, is set to store the name.

<111> Section 3.2.4.15: Windows implementations remove the Domain Admins group to the local Administrators group and remove the Domain Users group to the local users groups, as specified [MS-SAMR].

<112> Section 3.2.4.16: This method was added in Windows 2000 and is available in all subsequent versions.

<113> Section 3.2.4.16: Windows implementations ignore this parameter.

<114> Section 3.2.4.16: Windows RPC client machines set the *ServerName* parameter to a string that resolves to the IP protocol layer destination address of the RPC packet(s) it transmits. The string formats are as follows:

1. An IP address (for example, "10.0.0.23")

2. The fully qualified DNS name of the server

3. The NetBIOS name of the server

This RPC client behavior applies to all Windows RPC client machines, which includes Windows XP, Windows 2000, Windows 2000 Server, Windows Server 2003, Windows Vista, and Windows Server 2008.

<115> Section 3.2.4.16: 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 state different than it was in immediately prior to the call being processed.

<116> Section 3.2.4.16: Only on Windows XP SP2, Windows Server 2003 SP1, and Windows Vista, is the verification of the proper RPC protocol sequence enforced.

<117> Section 3.2.4.16: Only on Windows XP SP2, Windows Server 2003 SP1, and Windows Vista, is the verification of the proper RPC protocol sequence enforced.

<118> Section 3.2.4.16: Windows requires that the caller be a member of the Administrator group.

<119> Section 3.2.4.16: 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.

<120> Section 3.2.4.16: Windows implementations require a writable domain controller. For descriptions of a writable domain controller, see [MS-ADTS]. Windows implementations use the domain controller locator service specified in [MS-ADTS] to locate a writable domain controller.

<121> Section 3.2.4.16: Windows implementations require a writable domain controller. For descriptions of a writable domain controller, see [MS-ADTS]. Windows implementations use the domain controller locator service specified in [MS-ADTS] to locate a writable domain controller.

<122> Section 3.2.4.16: Windows implementations detect this condition when SamrSetInformationUser returns STATUS_USER_EXISTS or when an LDAP update to the samAccountName attribute on the DomainObject fails with the error LDAP_OBJECT_EXISTS. Windows implementations stop message processing and return NERR_UserExists in this case. Otherwise, message processing continues.

<123> Section 3.2.4.16: Windows populates the DNS suffix using Group Policy from the domain. Group Policy updates the following registry key:

- HKLM\Software\Policies\Microsoft\System\DNSclient NV PrimaryDnsSuffix

If this setting is not available, then the TCP/IP setting for the domain is queried and is used as the DNS suffix.

If that is not available either, then the DNS suffix of the DNS name of the domain is used as the DNS suffix.

<124> Section 3.2.4.16: Windows updates this attribute using the Security Account Manager (SAM) Remote Protocol (as specified in [MS-SAMR]). Windows uses a subset of RPC functions defined in this interface, and finally uses SamrSetInformationUser(2) to set the property.

<125> Section 3.2.4.16: Windows uses LDAP protocol to update **dnsHostName** and **servicePrincipalName** attributes. To be able to use LDAP protocol in this case, server implementation needs to know the DN of the object to modify. For this purpose, Windows uses IDL_DRSCrackNames (opnum 12) function in dsruapi interface as specified in [MS-DRSR].

<126> Section 3.2.4.16: Windows uses LDAP protocol to update **dnsHostName** and **servicePrincipalName** attributes. To be able to use LDAP protocol in this case, server implementation needs to know the DN of the object to modify. For this purpose, Windows uses IDL_DRSCrackNames (opnum 12) function in dsruapi interface as specified in [MS-DRSR].

<127> Section 3.2.4.16: Windows tries to update this property, but does not fail the operation if this modification fails.

<128> Section 3.2.4.16: Windows updates this attribute using the Security Account Manager (SAM) Remote Protocol (as specified in [MS-SAMR]). Windows uses a subset of RPC functions defined in this interface, and finally uses SamrSetInformationUser(2) to set the property.

<129> Section 3.2.4.16: Windows tries to update this property, but does not fail the operation if this modification fails.

<130> Section 3.2.4.17: This method was added in Windows 2000 and is available in all subsequent versions.

<131> Section 3.2.4.17: Windows implementations ignore this parameter.

<132> Section 3.2.4.17: Windows RPC client machines set the *ServerName* parameter to a string that resolves to the IP protocol layer destination address of the RPC packets it transmits. The string formats are as follows:

1. An IP address (for example, "10.0.0.23")

2. The fully qualified DNS name of the server

3. The NetBIOS name of the server

This RPC client behavior applies to all Windows RPC client machines, which includes Windows XP, Windows 2000, Windows 2000 Server, Windows Server 2003, Windows Vista, and Windows Server 2008.

<133> Section 3.2.4.17: Windows implementations ignore this parameter.

<134> Section 3.2.4.17: Windows implementations ignore this parameter.

<135> Section 3.2.4.17: Only on Windows XP SP2, Windows Server 2003 SP1, and Windows Vista, is the verification of the proper RPC protocol sequence enforced.

<136> Section 3.2.4.17: Only Windows XP SP2, Windows Vista, and Windows Server 2008 verify that the caller is local.

<137> Section 3.2.4.17: Windows requires that the caller be an authenticated user, and returns ERROR_ACCESS_DENIED if failed.

<138> Section 3.2.4.17: Windows implementation: The name is added as a NetBIOS group name. If the operation succeeds, the name is verified as valid; the name is deleted, as specified in [RFC1001], to undo the addition of the name. Otherwise, the name is not valid; error ERROR_INVALID_PARAMETER is returned.

<139> Section 3.2.4.17: 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 is deleted, as specified in [RFC1001], to undo the addition of the name. Otherwise, the name is not valid; error ERROR_DUP_NAME is returned if the name is already in use, as specified in [RFC1001].

<140> Section 3.2.4.17: Windows implementation: The domain locator protocol specified in [MS-ADTS], section 7.3.6, is used to verify that a domain controller can be found for the domain. If the domain locator succeeds finding any domain controller in the domain, this condition is verified. Otherwise, error ERROR_NO_SUCH_DOMAIN is returned.

<141> Section 3.2.4.17: Windows implementation: The domain locator protocol specified in [MS-ADTS] section 7.3.6 is used to determine if a domain controller can be found for the domain. If the domain locator finds a domain controller in the domain, error ERROR_DUP_NAME is returned. Otherwise, this condition is verified.

<142> Section 3.2.4.18: This method was added in Windows 2000 and is available in all subsequent versions.

<143> Section 3.2.4.18: Windows implementations ignore this parameter.

<144> Section 3.2.4.18: Windows RPC client machines set the *ServerName* parameter to a string that resolves to the IP protocol layer destination address of the RPC packets it transmits. The string formats are as follows:

1. An IP address (for example, "10.0.0.23")

2. The fully qualified DNS name of the server

3. The NetBIOS name of the server

This RPC client behavior applies to all Windows RPC client machines, which includes Windows XP, Windows 2000, Windows 2000 Server, Windows Server 2003, Windows Vista, and Windows Server 2008.

<145> Section 3.2.4.18: Only on Windows XP SP2, Windows Server 2003 SP1, and Windows Vista, is the verification of the proper RPC protocol sequence enforced.

<146> Section 3.2.4.18: Only Windows XP SP2, Windows Vista, and Windows Server 2008 verify that the caller is local.

<147> Section 3.2.4.18: Windows requires that the caller be a member of the authenticated users group.

<148> Section 3.2.4.18: On Windows XP SP2, Windows Vista, and Windows Server 2008 this call is failed on a domain controller. If the call fails, NERR_InvalidAPI is returned.

<149> Section 3.2.4.18: Windows implementations prefer a writable domain controller that has a computer account for the local server, that has secondary preference for any writable domain controller, and prefers the primary domain controller (PDC). For descriptions of a writable domain controller, computer account, and PDC, See [MS-ADTS]. Windows implementations use the domain controller locator service specified in [MS-ADTS] and [MS-NRPC] to locate the preferred domain controller.

<150> Section 3.2.4.18: Windows implementation uses the LDAP protocol specified in [RFC2251] to perform this query. On Windows 2000, the security provider used for the security context for the query is the Security Support Provider Interface (SSPI). For information about SSPI, see [RFC1964]. On Windows Server 2003 and Windows Vista, the security provider used for the security context for the query is negotiated with SPNEGO, as specified in [MS-SPNG].

<151> Section 3.2.4.19: This method was added in Windows XP and is available in all subsequent versions.

<152> Section 3.2.4.19: Windows implementations ignore this parameter.

<153> Section 3.2.4.19: Windows RPC client machines set the *ServerName* parameter to a string that resolves to the IP protocol layer destination address of the RPC packets it transmits. The string formats are as follows:

1. An IP address (for example, "10.0.0.23")

2. The fully qualified DNS name of the server

3. The NetBIOS name of the server

This RPC client behavior applies to all Windows RPC client machines, which includes Windows XP, Windows 2000, Windows 2000 Server, Windows Server 2003, Windows Vista, and Windows Server 2008.

<154> Section 3.2.4.19: Windows implementations always pass Reserved.

Note The value of Reserved is 0x00000000.

<155> Section 3.2.4.19: Windows Vista and later: Indicates that the method SHOULD prevent concurrent calls.

<156> 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 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.

<157> Section 3.2.4.19: Only Windows XP SP2, Windows Server 2003 SP1, and Windows Vista verify that the proper RPC protocol sequence is enforced.

<158> Section 3.2.4.19: Windows requires that the caller be a member of the Administrators group. The error ERROR_ACCESS_DENIED is returned if this condition is not met.

<159> Section 3.2.4.19: Windows XP and Windows Vista return ERROR_NOT_SUPPORTED if this method is invoked.

<160> Section 3.2.4.19: 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.

<161> Section 3.2.4.19: The Windows implementations require a writable domain controller. If the server is an RODC, then Windows implementations require a Windows Server 2008 writable domain controller. See [MS-ADTS] for descriptions of a writable domain controller. Windows implementation uses the domain controller locator service specified in [MS-ADTS] to locate a writable domain controller.

<162> Section 3.2.4.19: Windows uses the LDAP protocol to update the **msDS-AdditionalDnsHostName** attribute. To be able to use the LDAP protocol in this case, a server implementation needs to know the domain name of the object to modify. For this purpose, Windows uses the IDL_DRSCrackNames (opnum 12) function in the dsruapi interface as specified in [MS-DRSR].

<163> Section 3.2.4.20: This method was added in Windows XP and is available in all subsequent versions.

<164> Section 3.2.4.20: Windows implementations ignore this parameter.

<165> Section 3.2.4.20: Windows RPC client machines set the *ServerName* parameter to a string that resolves to the IP protocol layer destination address of the RPC packets it transmits. The string formats are the following:

1. An IP address (for example, "10.0.0.23");

2. The fully qualified DNS name of the server; and

3. The NetBIOS name of the server.

This RPC client behavior applies to all Windows RPC client machines, which includes Windows XP, Windows 2000, Windows 2000 Server, Windows Server 2003, Windows Vista, and Windows Server 2008.

<166> Section 3.2.4.20: Windows implementations always pass *Reserved*.

Note The value of Reserved is 0x00000000.

<167> Section 3.2.4.20: Windows Vista and later: Indicates that the method SHOULD prevent concurrent calls.

<168> 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.

<169> Section 3.2.4.20: Only on Windows XP SP2, Windows Server 2003 SP1, and Windows Vista is the verification of the proper RPC protocol sequence enforced.

<170> Section 3.2.4.20: Windows requires that the caller must be a member of the administrator group. The error **ERROR_ACCESS_DENIED** is returned if this condition is not met.

<171> Section 3.2.4.20: Windows XP and Windows Vista return **ERROR_NOT_SUPPORTED** if this method is invoked.

<172> Section 3.2.4.20: 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.

<173> Section 3.2.4.20: Windows implementations require a writable domain controller. If the server is an RODC, then Windows implementations require a Windows Server 2008 writable domain controller. For descriptions of a writable domain controller, see [MS-ADTS]. Windows implementations use the Domain Controller Locator service specified in [MS-ADTS] to locate a writable domain controller.

<174> Section 3.2.4.20: Windows uses the LDAP protocol to update the **msDS-AdditionalDnsHostName** attribute. To be able to use the LDAP protocol in this case, the server implementation needs to know the domain name of the object to modify. For this purpose, Windows uses the **IDL_DRSCrackNames (opnum 12)** function in the **dsruapi** interface, as specified in [MS-DRSR].

<175> Section 3.2.4.21: This method was added in Windows XP and is available in all subsequent versions.

<176> Section 3.2.4.21: Windows implementations ignore this parameter.

<177> Section 3.2.4.21: Windows RPC client machines set the *ServerName* parameter to a string that resolves to the IP protocol layer destination address of the RPC packets it transmits. The string formats are as follows:

1. An IP address (for example, "10.0.0.23")

2. The fully qualified DNS name of the server

3. The NetBIOS name of the server

This RPC client behavior applies to all Windows RPC client machines, which includes Windows XP, Windows 2000, Windows 2000 Server, Windows Server 2003, Windows Vista, and Windows Server 2008.

<178> Section 3.2.4.21: Windows implementations always pass Reserved.

Note The value of Reserved is 0x00000000.

<179> Section 3.2.4.21: Windows Vista and later: Indicates that the method SHOULD prevent concurrent calls.

<180> Section 3.2.4.21: 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.

<181> Section 3.2.4.21: Only Windows XP SP2, Windows Server 2003 SP1, and Windows Vista enforce verification of the proper RPC protocol sequence.

<182> Section 3.2.4.21: Windows requires that the caller be a member of the Administrators group. The error ERROR_ACCESS_DENIED is returned if this condition is not met.

<183> Section 3.2.4.21: Windows XP and Windows Vista return ERROR_NOT_SUPPORTED if this method is invoked.

<184> Section 3.2.4.21: 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.

<185> Section 3.2.4.21: Windows implementation requires a writable domain controller. If the server is an RODC, then Windows implementations require a Windows Server 2008 writable domain controller. For descriptions of a writable domain controller, see [MS-ADTS]. Windows implementations use the domain controller locator service, as specified in [MS-ADTS], to locate a writable domain controller.

<186> Section 3.2.4.21: Windows uses the LDAP protocol to update the **msDS-AdditionalDnsHostName** attribute. To be able to use the LDAP protocol in this case, the server implementation needs to know the domain name of the object to modify. For this purpose, Windows uses IDL_DRSCrackNames (opnum 12) function in dsruapi interface as specified in [MS-DRSR].

<187> Section 3.2.4.21: Windows implementations perform the **SamAccountName** update using a sequence that includes **SamrOpenUser**, **SamrQueryInformationUser**, and **SamrSetInformationUser**, rather than LDAP. The transport for these RPC calls requires the server to establish an authenticated SMB (as specified in an [MS-RPCE]) session between the server and *DomainController*. The identity established is that of *DomainAccount*, and the credentials supplied during authentication are those specified in *EncryptedPassword*. This identity is used by *DomainController* for the purpose of performing access checks on all operations targeting *DomainController* over this transport, as specified in [MS-SAMR].

<188> Section 3.2.4.22: This method was added in Windows XP and is available in all subsequent versions.

<189> Section 3.2.4.22: Windows implementations always pass Reserved.

Note The value of Reserved is 0x00000000.

<190> Section 3.2.4.22: Only on Windows XP SP2, Windows Server 2003 SP1, and Windows Vista, is the verification of the proper RPC protocol sequence enforced.

<191> Section 3.2.4.22: Windows requires that the caller be a member of the Administrators group. The error ERROR_ACCESS_DENIED is returned if this condition is not met.

<192> Section 3.2.4.22: Windows XP and Windows Vista return ERROR_NOT_SUPPORTED if this method is invoked.

8 Index