[MS-SMB]:
Server Message Block (SMB) Protocol Specification

Revision Summary

Date	Revision History	Revision Class	Comments
04/03/2007	0.01		MCPP Milestone Longhorn Initial Availability
07/03/2007	1.0	Major	MLonghorn+90
07/20/2007	2.0	Major	Updated and revised the technical content.
08/10/2007	3.0	Major	Updated and revised the technical content.
09/28/2007	4.0	Major	Updated and revised the technical content.

Date	Revision History	Revision Class	Comments
10/23/2007	5.0	Major	Updated and revised the technical content.
11/30/2007	5.0.1	Editorial	Revised and edited the technical content.
01/25/2008	5.0.2	Editorial	Revised and edited the technical content.

Table of Contents

1 Introduction

This document defines extensions to the existing Common Internet File System (CIFS), as specified in [CIFS], that have been implemented by Microsoft since the publication of the [CIFS] specification. This document also defines Windows behavior with respect to optional behavior in the base specification.

1.1 Glossary

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

@GMT Token
8.3 Name
ASCII
Blocking Mode (of a Named Pipe)
Copychunk Resume Key
Distributed File System (DFS)
Domain
Fid
File System Control (FSCTL)
Globally Unique Identifier (GUID)
Guest Account
Information Level
Little-Endian
Mailslot
Maximum Transmission Unit (MTU)
Message Mode
Multiplexed Request
Named Pipe
Network Byte Order
NTFS file system
Oplock
Oplock Break
OEM Character
Pipe Instance
Pipe State
Process Identifier
Raw Read (on a Named Pipe)
Raw Write (on a Named Pipe)
Reparse Point
Security Identifier (SID)
Session
Share
Share Connect
SMB Connection
SMB Dialect
SMB Session
Snapshot
Stream
Unicode
UTC
Virtual Connection
Volume Identifier

The following terms are specific to this document:

FileId: A 64-bit value used to represent a file. The value of a FileId is unique on a single volume of a local file system or a remote file server. A FileId is not guaranteed to be unique across volumes, but the file system on the server MUST guarantee that it is unique within a given volume if FileIds are supported. FileIds are not supported by all local file systems. On Windows, the **NTFS** file system supports FileIds, but the FAT file system does not support them.

Client: All the references to client in this specification refer to SMB Client unless explicitly stated otherwise.

Server: All the references to server in this specification refer to SMB Server unless explicitly stated otherwise.

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

1.2 References

1.2.1 Normative References

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

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

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

[IANAPORT] Internet Assigned Numbers Authority, "Port Numbers", November 2006, http://www.iana.org/assignments/port-numbers

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

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

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

[MS-FSCC] Microsoft Corporation, "File System Control Codes", July 2007.

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

[MS-KILE] Microsoft Corporation, "Kerberos Protocol Extensions", January 2007.

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

[MS-RAP] Microsoft Corporation, "Remote Administration Protocol Specification", March 2007.

[NETBEUI] IBM Corporation, "LAN Technical Reference: 802.2 and NetBIOS APIs", 1986, http://publibz.boulder.ibm.com/cgi-bin/bookmgr_OS390/BOOKS/BK8P7001/CCONTENTS

If you have any trouble finding [NETBEUI], 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

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

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

[RFC2104] Krawczyk, H., Bellare, M., and Canetti, R., "HMAC: Keyed-Hashing for Message Authentication", RFC 2104, February 1997, http://www.ietf.org/rfc/rfc2104.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

[RFC2743] Linn, J., "Generic Security Service Application Program Interface Version 2, Update 1", RFC 2743, January 2000, http://www.ietf.org/rfc/rfc2743.txt

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

[XOPEN-SMB] The Open Group, "Protocols for X/Open PC Interworking: SMB, Version 2", The Open Group, 1992, ISBN: 1872630456.

If you have any trouble finding [XOPEN-SMB], please check here.

1.2.2 Informative References

[IPX] Microsoft Corporation, "Internetwork Packet Exchange (IPX)", http://msdn2.microsoft.com/en-us/library/ms817906.aspx

[MD5Collision] Klima, V., "Tunnels in Hash Functions: MD5 Collisions Within a Minute", March 2006, http://eprint.iacr.org/2006/105.pdf

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

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

[MS-MAIL] Microsoft Corporation, "Remote Mailslot Protocol Specification", March 2007.

[MS-MSRP] Microsoft Corporation, "Messenger Service Remote Protocol Specification", August 2007.

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

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

[MS-SRVS] Microsoft Corporation, "Server Service Remote Protocol Specification", January 2007.

[MSBRWSE] Thompson IV, D. and McLaughlin, R., "MS Windows NT Browser", https://www.microsoft.com/technet/archive/winntas/deploy/prodspecs/ntbrowse.mspx

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

[MSLOT] Microsoft Corporation, "Mailslots", http://msdn2.microsoft.com/en-us/library/aa365576.aspx

[NBF] Microsoft Corporation, "Comparison of Windows NT Network Protocols", November 2006, http://support.microsoft.com/kb/q128233

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

[NWLINK] Microsoft Corporation, "Description of Microsoft NWLINK IPX/SPX-Compatible Transport", October 2006, http://support.microsoft.com/?kbid=203051

[RAP] Leach, P. and Naik, D., "CIFS Remote Administration Protocol–Preliminary Draft", February 1997, ftp://ftp.microsoft.com/developr/drg/CIFS/cifsrap2.txt

[RFC894] Hornig, C., "A Standard for the Transmission of IP Datagrams over Ethernet Networks", RFC 894 April 1984, http://www.ietf.org/rfc/rfc894.txt

[RYAN] Ryan, R. and Ryan, B., "LAN Manager: A Programmer's Guide, Version 2", Microsoft Press, July 1990, ISBN: 1556151667.

[SNIA] Storage Networking Industry Association, "Common Internet File System (CIFS) Technical Reference, Revision 1.0", March 2002, http://www.snia.org/tech_activities/CIFS/CIFS-TR-1p00_FINAL.pdf

1.3 Protocol Overview (Synopsis)

Client systems use the Common Internet File System (CIFS) Protocol to request file and print services from **server** systems over a network. CIFS is a stateful protocol, where clients establish a **session** with a server and use that session for a variety of requests to access files, printers, and interprocess communication (IPC) mechanisms such as **named pipes**. CIFS imposes state to maintain an authentication context, cryptographic operations, file semantics such as locking, and similar features. A detailed overview of how the CIFS Protocol works is as specified in [CIFS] section 2.

The Server Message Block (SMB) Protocol extends the CIFS Protocol with additional security, file, and disk management support. These extensions do not alter the basic message sequencing of the CIFS Protocol but introduce new flags, extended requests and responses, and new **information levels**. All of these extensions follow a request/response pattern in which the client initiates all requests. The base protocol allows for one exception to this pattern—**oplock breaks**—as specified in [CIFS] section 2.7.

This document defines the SMB Protocol extensions to CIFS that provide support for the following features:

- New authentication methods, including Kerberos. The Negotiate and Session Setup commands have been enhanced to carry opaque security tokens to support mechanisms compatible with the Generic Security Service (GSS).
- Message signing to guarantee packet integrity. The SMB message header now optionally contains a cryptographic checksum of the contents of an SMB message, preventing tampering of the message while in transit.

- Enumeration and access to previous versions of files. A new subcommand that uses a **file system control (FSCTL)** allows the client to query the server for the presence of older versions of files. If the server implements a file system with versioning, this can be exposed to clients.
- Client requests for server-side data movement operations between files without requiring the data to be read by the client and then written back to the server. As specified in [CIFS], to copy a file on the server requires the client to read all the data from the server and then write the data back to the server. SMB introduces a method by which such an operation can be done entirely on the server without consuming network resources.
- **SMB connections** that use direct TCP for the SMB transport without the use of NetBIOS over TCP/UDP, as specified in [RFC1001] and [RFC1002]. CIFS specifies the use of NetBIOS over TCP/UDP for connections, as specified in [CIFS] section 2.5. SMB includes a method to connect directly over TCP without involving NetBIOS. Information about NetBIOS is specified in [NETBEUI].
- SMB connections that use NetBIOS over IPX for the SMB transport.
- SMB connections that use NetBIOS Extended User Interface (NetBEUI), as specified in [NETBEUI], for the SMB transport.
- Support for the use of native file and file system information levels in query and set operations. SMB allows for FSCTLs to be sent from clients to servers to interrogate or request operations of the underlying file system exposed by the server.
- Support for retrieving extended information in response to **share connect** and file open operations. Certain server functionality and indicators (such as the need for the client to cache the contents of a **share**) are new in SMB and are returned to the client through these extensions to existing commands.

Many of these capabilities are exposed in enhancements to the SMB_COM_NEGOTIATE and SMB_COM_SESSION_SETUP_ANDX commands requests and responses (as specified in sections SMB_COM_NEGOTIATE Client Request Extension (section 2.2.2), SMB_COM_NEGOTIATE Server Response Extension (Extended Security) (section 2.2.3), SMB_COM_SESSION_SETUP_ANDX Client Request Extension (section 2.2.4), and SMB_COM_SESSION_SETUP_ANDX Server Response Extension (section 2.2.5)).

1.4 Relationship to Other Protocols

The extensions to CIFS rely on SPNEGO (as specified in [RFC4178]) for authentication, which in turn MAY rely on Kerberos (as specified in [MS-KILE]) and/or the NT LAN Manager (NTLM) (as specified in [MS-NLMP]) challenge/response authentication protocol.

The Server Message Block (SMB) Version 2.0 Protocol is a new version of SMB. For more information about SMB Version 2.0 Protocol, see [MS-SMB2]. This specification does not require implementation of the Server Message Block (SMB) Version 2.0 Protocol protocol.

The following protocols extend this specification to provide additional functionality:

- The Distributed File System (DFS) Referral Protocol, as specified in [MS-DFSC]. For more information, see [MSDFS]. For management of **DFS**, see [MS-DFSNM].
- The Remote Administration Protocol, as specified in [MS-RAP]. For more information, see [RAP] and [XOPEN-SMB].

The following protocols can use SMB as a transport:

- The Remote Procedure Call Protocol Extensions. Note that when named pipes are used, this protocol requires the SMB Protocol. For more information, see [MS-RPCE].
- The Remote Mailslot Protocol. This protocol can use SMB as a transport, but supports other transports as well. For more information, see [MS-MAIL].
- The Common Internet File System (CIFS) Browser Protocol. This protocol uses the Remote Mailslot Protocol and the Remote Administration Protocol as transport protocols, which in turn can utilize this specification. It does not utilize this specification directly, but is included here for completeness. For more information, see [MS-BRWS] and [MSBRWSE].

1.5 Prerequisites/Preconditions

The extensions specified in this document require that a client that initiates a CIFS connection MUST negotiate an **SMB dialect** of NTLM 0.12 (as specified in [CIFS] section 2.5). Detailed prerequisites related to server name determination and resolution are specified in [CIFS] sections 2.1 and 2.2.

1.6 Applicability Statement

The extensions specified in this document are applicable to environments in which the security characteristics of the base protocol, as specified in [CIFS], are insufficient. In particular, these extensions provide for enhanced message integrity and stronger authentication mechanisms, and are thus applicable to environments that require these features.

The extensions are applicable to an environment that requires tighter data retention policies. In particular, through the use of previous version capabilities, the extensions allow access to versions of a file that have been changed or deleted when the server supports this capability. This feature is applicable to environments that require more stringent data retention policies that include maintaining access to previous versions of files.

1.7 Versioning and Capability Negotiation

This document covers versioning issues in the following areas:

- Supported Transports: The extensions in this document add additional transports, as defined in section 2.1.
- Security and Authentication Methods: The extensions in this document add additional authentication methods, as specified in section 3.2.4.2.3.
- Capability Negotiation: The extensions in this document use capability negotiation, as specified in [CIFS] sections 2.5 and 4.1.1.

SMB dialect negotiation is handled as specified in [CIFS] sections 2.5 and 4.1.1. The extensions specified in this document introduce no new dialects and apply only to connections that MUST have negotiated the NTLM 0.12 dialect. The extensions specified in this document are detected via the following methods:

- They can be returned in the **Capabilities** field, as specified in [CIFS] section 4.1.1. Specific new capability options are defined in this document.
- They can be supplied or returned in the **Flags** and **Flags2** fields of the SMB header, as specified in [CIFS] sections 3.1.1 and 3.1.2.
- A server can return an error code (STATUS_NOT_SUPPORTED) when a client request is sent to a server for a new feature that is not supported.

A client written to support these extensions MUST NOT require that the target server has implemented these extensions to successfully connect. Thus, a server that does not implement an extension is still accessible by a client that implements that extension, although the relevant new features may not be available. The one exception is that a client MAY be offer the capability to be configured to require the new security features to create a more secure environment, so that the client could restrict itself from connecting successfully to servers that do not implement these features.

Negotiation of the use of the Generic Security Service Application Program Interface (GSS API) for authentication is specified in section 3.2.5.2. The GSS API is specified in [RFC4178].

Negotiation of the use of message signing is specified in section 3.2.5.2.

1.8 Vendor-Extensible Fields

Since the publication of [CIFS], several fields and commands have been reserved as CIFS extensions for UNIX, UNIX-like systems, and Apple Macintosh systems. These UNIX and Macintosh extensions are not considered part of the extensions specified in this document. For more information, see [SNIA] section 4.1 and Appendix D, "CIFS UNIX Extension". Further information is specified in [XOPEN-SMB] section 5.2, "SMB Command Codes".

There are several vendor-extensible fields that MUST be provided by the server to implement the CIFS Protocol with the following extensions:

- **Fid** (SMB file identifier)
- **Copychunk resume key**

There are several vendor-extensible fields that MAY be provided by the server to implement the CIFS Protocol with the following extensions:

- **FileId** (file identifier)
- VolumeId (**volume identifier**)

1.8.1 Algorithm for Fid Generation

The vendor MUST select or create an algorithm for the generation of SMB file identifiers (Fids) on the SMB server. Any algorithm MAY be used as long as the following conditions are met. The generation of Fids on the SMB server MUST satisfy the following constraints:

- The Fid MUST be a 16-bit opaque value generated by an SMB server for a specific create or open request.
- The Fid MUST be unique on the SMB server for a given create or open request.
- The Fid MUST be unique on the SMB server for a given SMB connection.
- The Fid MUST remain valid for the lifetime of the SMB connection on which the open request is performed, or until the client sends a request to the server to close the Fid.
- When a Fid is closed, the value MAY be reused for another create or open request.
- A Fid equal to 0xffff MUST be treated as invalid. All other possible values for Fid are valid. The value 0xffff is used (as specified in [CIFS]) as well as this protocol to specify all Fids or no Fid, depending on the context in which it is used.

1.8.2 Algorithm for FileId Generation

The vendor MAY select or create an algorithm for the generation of file identifiers (FileIds) on the SMB server. Any algorithm MAY be used for generating these identifiers as long as the following conditions are met. The generation of FileId on the SMB server MUST satisfy the following constraints:

- The FileId MUST be a 64-bit opaque value.
- The FileId MUST be unique for a file on a given volume.
- The FileId for a file MUST persist for the lifetime of a file on a volume. A FileId MUST NOT be changed when a file is renamed. When the file is deleted, the FileId MAY be reused.
- All possible values for FileId are valid.

1.8.3 Algorithm for VolumeId Generation

The vendor MAY select or create an algorithm for the generation of volume identifiers (VolumeIds) on the SMB server. Any algorithm MAY be used for generating these identifiers as long as the following conditions are met. The generation of VolumeIds on the SMB server MUST satisfy the following constraints:

- The VolumeId MUST be a 128-bit opaque value.
- The VolumeId MUST be unique for a logical file system volume on a given server.
- The VolumeId for the volume MAY change while the system is running. The VolumeId MAY change when the system is restarted.
- All possible values for the VolumeId are valid.

1.8.4 Algorithm for Copychunk Resume Key Generation

The vendor MUST select or create an algorithm for the generation of the Copychunk Resume Key on the SMB server. Any algorithm MAY be used for generating the Copychunk Resume Key as long as the following conditions are met. The generation of Copychunk Resume Key on the SMB server MUST satisfy the following constraints:

- The Copychunk Resume Key MUST be a 24-byte opaque value generated by an SMB server in response to a request by the client (an SMB_COM_NT_TRANSACTION request with an NT_TRANSACT_IOCTL subcommand for the FSCTL_SRV_REQUEST_RESUME_KEY). See section 2.2.14.7.2.
- The Copychunk Resume Key MUST be unique on the SMB server for a given open file on a server.
- The Copychunk Resume Key MUST remain valid for the lifetime of the open file on the server, or until the file is closed.
- All possible values for the Copychunk Resume Key are valid.

1.9 Standards Assignments

In addition to any standards assignments specified in [CIFS], the direct TCP SMB transport, as specified in section 2.1 uses the following assignment.

Parameter	TCP port value	Reference
Microsoft-DS	445 (0x01BD)	[IANAPORT]

In addition to any standards assignments specified in [CIFS] the NetBIOS over IPX SMB transport, as specified in section 2.1, uses the following assignment.

Parameter	IPX socket value	Reference
NetBIOS socket	1109 (0x0455)	[IPX]

2 Messages

The following sections specify how Server Message Block (SMB) Protocol messages are transported and specify SMB Protocol message syntax.

2.1 Transport

In addition to the NetBIOS over TCP/UDP transport (as specified in [RFC1001], [RFC1002], and [CIFS] section 2.5), the extended version of the protocol supports the use of the following other transports for SMB messages:

- Direct TCP
- NetBIOS over IPX
- NetBIOS Extended User Interface (NetBEUI)

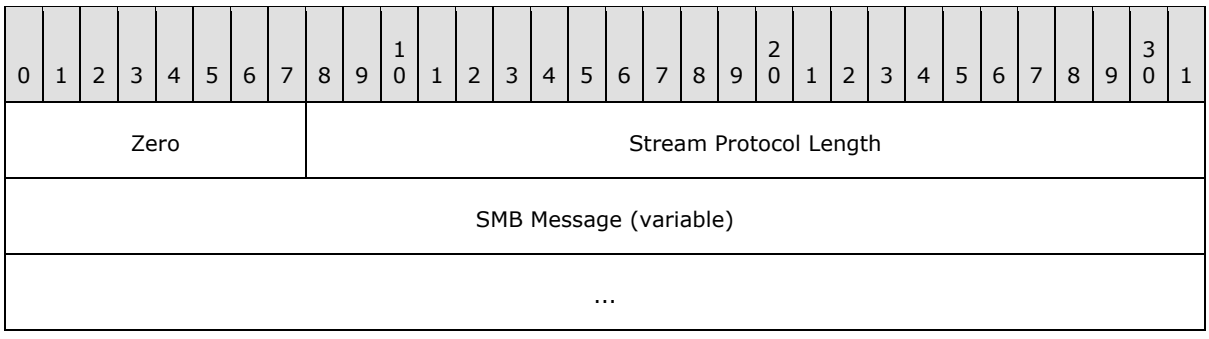
2.1.1 Direct TCP Transport

The extended version of the SMB Protocol MAY use direct TCP over either IPv4 or IPv6 as a reliable **stream**-oriented transport for SMB messages. No NetBIOS layer is provided or used. TCP provides a full, duplex, sequenced, and reliable transport for the connection. When using TCP as the reliable connection-oriented transport, the extended version of the SMB Protocol makes no higher-level attempts to ensure sequenced delivery of messages between a client and server. The TCP transport has mechanisms to detect failures of either the client node or server node and to deliver such an indication to the client or server software so that it can clean up the state.

When using direct TCP as the SMB transport, the implementer MUST establish a TCP connection from an SMB client to a TCP port on the server. The TCP source port used by the SMB client MAY be any TCP port value. The SMB server SHOULD listen for connections on port 445. This port number has been registered with the IANA and has been officially assigned for Microsoft-DS.<1>

This transport supports only the SMB file server functionality. The support of **mailslot** messages is not required for SMB file server functionality.

When using direct TCP as the SMB transport, you MUST prepend a four-byte Direct TCP Transport to each SMB message. This Direct TCP Transport header MUST be formatted as a byte of zero (eight zero bits) followed by three bytes that indicate the length of the SMB message that is encapsulated. The body of the SMB packet follows as a variable-length payload. A Direct TCP Transport packet is displayed below as it appears over the network (in **network byte order**) with the SMB message.



Zero (1 byte): First byte of the Direct TCP Transport packet header MUST be zero (0x00).

Stream Protocol Length (3 bytes): The length, in bytes, of the SMB message. This length is formatted as a three-byte integer in network byte order. The length field does not include the four-byte Direct TCP Transport header; rather, it is only the length of the enclosed SMB message.

SMB Message (variable): The body of the SMB packet. The length of an SMB message varies based on the SMB command that the message represents.<2>

2.1.2 NetBIOS Over IPX Transport

The extended version of the SMB Protocol MAY use NetBIOS over IPX (NBIPX) as a transport for SMB messages. The NetBIOS session service provides a reliable message-oriented transport for use over IPX. When using NetBIOS to provide a reliable message-oriented transport over IPX, the SMB Protocol makes no higher-level attempts to ensure reliable sequenced delivery of messages between the client and server. The NetBIOS session service has mechanisms to detect failures of either the client node or server node and to deliver such an indication to the client or server software so that they can clean up the state.

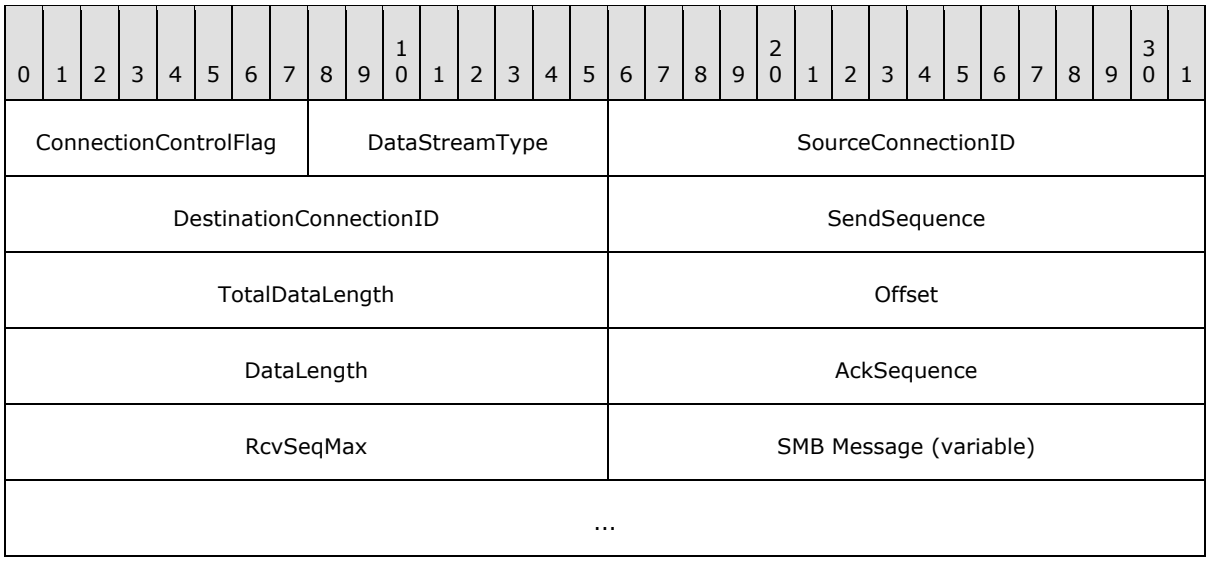
Novell introduced an implementation of NetBIOS over IPX in 1986. The Novell implementation uses IPX datagrams to carry the NetBIOS packets. Novell defined IPX socket 0x455 as the socket to be used for NetBIOS IPX packets.

When using NetBIOS over IPX as the SMB transport, the implementer MUST establish an IPX connection from an SMB client to an IPX socket on the server. The IPX source socket used by the SMB client MUST be set to the IPX NetBIOS socket 0x455, the value defined by Novell for NetBIOS. The IPX destination socket (that the SMB client sends requests to) MUST be set to the IPX NetBIOS socket, 0x455. The SMB server MUST listen for requests on the IPX NetBIOS socket 0x455. The IPX source socket used by the SMB server for sending responses MUST be set to the IPX NetBIOS socket 0x455. The IPX destination socket used by the SMB server for responses MUST be set to the IPX NetBIOS socket 0x455.

When using NetBIOS over IPX as the SMB transport, an IPX packet header plus an IPX NetBIOS session packet header MUST be prepended to each SMB message. Both the IPX packet header (for more information, see [IPX]) and the IPX NetBIOS session packet header (for more information, see [NWLINK]) are defined by Novell (for more information, see [NOVELL]). The body of the SMB packet follows as a variable-length payload.

The Microsoft implementation of NetBIOS over IPX supports an extension to Novell NetBIOS over IPX called NWNBLink NetBIOS. This extension is defined in the extensions to [CIFS] because this is the format used between Windows-based SMB clients and servers when using NetBIOS over IPX as the SMB transport.

The IPX NetBIOS session packet that contains the SMB message is displayed below as it appears over the network (in network byte order).



ConnectionControlFlag (1 byte): An 8-bit field that contains connection control flags. The bits in this field provide control information to NetBIOS session packets. Specific combinations of the following bits that MUST be used are as follows.

Value	Meaning
NB_CONTROL_NEW_NB 0x01	Identifies the Microsoft NWNBLink implementation of NetBIOS over IPX. This bit field MUST be added on the session initialize frame by the sender and on the corresponding session acknowledgment response frame sent by the receiver to indicate the use of NWNBLink NetBIOS over IPX and multiple frame windowing instead of Novell NetBIOS over IPX. This bit field MUST NOT be set on any other NetBIOS over IPX frames that use the Microsoft NWNBLink implementation.
NB_CONTROL_RESEND 0x08	Indicates on the acknowledgment sent by the receiver that the receiver is requesting the sender to resend a packet.
NB_CONTROL_EOM 0x10	Indicates the end of a message.
NB_CONTROL_ATTENTION 0x20	Indicates an attention. This flag is not used by NWNBLink NetBIOS. This flag is not set when a frame is sent and is ignored when a packet is received.
NB_CONTROL_SEND_ACK 0x40	Indicates a send acknowledgment.
NB_CONTROL_SYSTEM 0x80	Indicates a system packet.

DataStreamType (1 byte): An 8-bit unsigned field that indicates the type of data contained within the frame. This field MUST be set to one of the following values.

Value	Meaning
NB_CMD_SESSION_DATA	Session data. This value MUST be used for the session initialize,

Value	Meaning
0x06	session acknowledgment, and session data frames.
NB_CMD_SESSION_END 0x07	Session end. This value MUST be used for session-end frames.
NB_CMD_SESSION_END_ACK 0x08	Session-end acknowledgment. This value MUST be used for session-end acknowledgment frames to indicate the receipt of a session-end frame.

SourceConnectionID (2 bytes): A 16-bit unsigned number in **little-endian** byte order that MUST be set to the connection ID number of the source station. This field is assigned by the source station and is used to multiplex communications so that current active SMB connections can use the same IPX socket number.

DestinationConnectionID (2 bytes): A 16-bit unsigned number in little-endian byte order that MUST be set to the connection ID number of the destination station. This field MUST be assigned by the destination station to a value other than 0xffff. This field MAY be used to multiplex communications so that current active SMB connections can use the same IPX socket number. During the initial SMB connection establishment, this field MUST be set to 0xffff because the sender does not yet know the destination connection ID the receiver will use.

SendSequence (2 bytes): A 16-bit unsigned number in little-endian byte order that MUST be set by the sender to the sequential frame number of the frame being sent. The sequential frame number MUST be incremented for each new NetBIOS session frame that is sent, but not for a frame that is being resent. The value in the first packet sent by the sender on the session initialization frame MUST be set to 0.

TotalDataLength (2 bytes): A 16-bit unsigned number in little-endian byte order that MUST be set to the total length, in bytes, of this NetBIOS session message. The maximum packet size for some IPX routers is 576 bytes, including the IPX header. Because of this, the client MAY need to send an SMB over NBIPX message split over multiple NBIPX packets.

Offset (2 bytes): If an SMB message is sent in multiple NBIPX packets, this 16-bit unsigned number in little-endian byte order MUST be set to the offset from the start of the NetBIOS message to the data in this message. If the SMB message is sent in a single NBIPX packet, this field MUST be set to 0.

DataLength (2 bytes): A 16-bit unsigned number in little-endian byte order that MUST be set to the length, in bytes, of the part of the SMB message contained in this packet.

AckSequence (2 bytes): A 16-bit unsigned number in little-endian byte order that MUST be set by the receiver to the next frame number that it expects to receive. This tells the sender which packets have been accepted by the receiver.

RcvSeqMax (2 bytes): A 16-bit unsigned number in little-endian byte order that MUST be set by the receiver to the highest frame number that the sender MAY send.

SMB Message (variable): A variable length field that contains the body of the SMB packet.<3>

Novell NetBIOS uses two frames to initialize a session. The initiator of a session sends the session initialize frame with the **ConnectionControlFlag** field set to NB_CONTROL_SEND_ACK. The responder sends back a session initialize acknowledgment frame with the **ConnectionControlFlag** field set to NB_CONTROL_SYSTEM.

NWNBLink NetBIOS use two frames to initialize a session, but modifies the flag bits that are set in the **ConnectionControlFlag** field. The initiator of a NWNBLink NetBIOS session MUST send the session initialize frame with the **ConnectionControlFlag** field set to NB_CONTROL_SEND_ACK combined with NB_CONTROL_NEW_NB and the **DataStreamType** field MUST be set to NB_CMD_SESSION. A responder that supports NWNBLink NetBIOS MUST send back a session initialize acknowledgment frame with the **ConnectionControlFlag** field set to NB_CONTROL_SYSTEM combined with NB_CONTROL_NEW_NB and the **DataStreamType** field MUST be set to NB_CMD_SESSION.

After a session is initialized, all session data frames are exchanged with the **DataStreamType** field that MUST be set to NB_CMD_SESSION. A sender MAY set the **ConnectionControlFlag** field to NB_CONTROL_EOM combined with NB_CONTROL_SEND_ACK on a session data frame that is the end of an SMB message to request an acknowledgment from the receiver.

For a normal acknowledgment of a session data frame, the **ConnectionControlFlag** field MUST be set to NB_CONTROL_SYSTEM. When a resend is requested on the acknowledgment, the **ConnectionControlFlag** field MUST be set to NB_CONTROL_SYSTEM combined with NB_CONTROL_RESEND. When an acknowledgment query is being sent on the acknowledgment, the **ConnectionControlFlag** field MUST be set to NB_CONTROL_SYSTEM combined with NB_CONTROL_SEND_ACK.

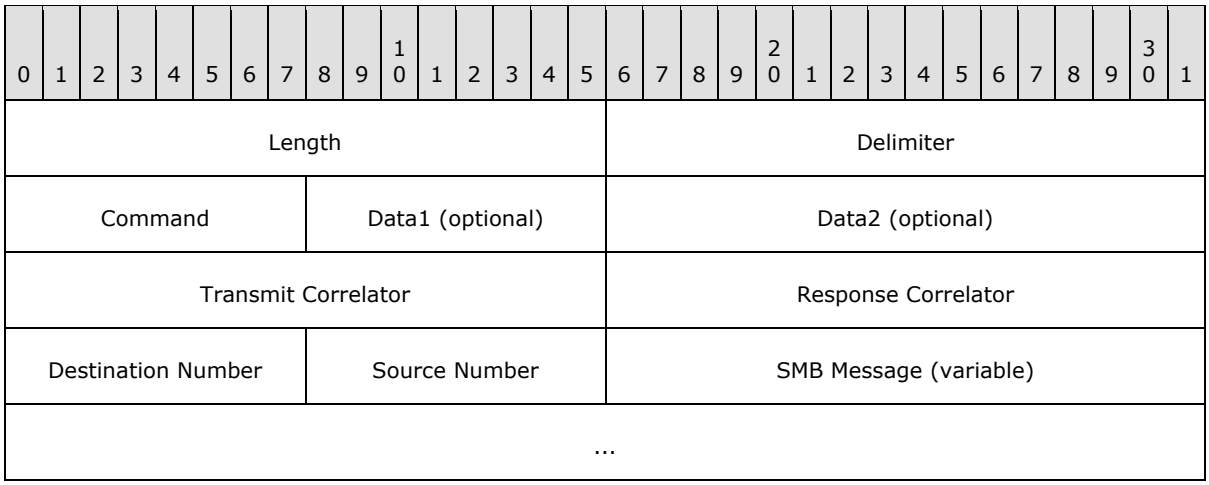
A NWNBLink NetBIOS session is ended when the initiator or responder sends a session end frame with the **ConnectionControlFlag** field that MUST be set to NB_CONTROL_SEND_ACK and the **DataStreamType** field that MUST be set to NB_CMD_SESSION_END. The receiver of the session end frame responds with a session end acknowledgment frame with the **DataStreamType** field that MUST be set to NB_CMD_SESSION_END_ACK and the **ConnectionControlFlag** field that MUST be set to 0.

2.1.3 NetBIOS Extended User Interface Transport

The extended version of the SMB Protocol MAY use NetBIOS Extended User Interface (NetBEUI) 3.0 as a transport for SMB messages. The NetBIOS session service provides a reliable message-oriented transport. When using NetBIOS to provide a reliable message-oriented transport using NetBEUI, the SMB Protocol makes no higher-level attempts to ensure reliable, sequenced delivery of messages between the client and server. The NetBIOS session service on NetBEUI has mechanisms to detect failures of either the client node or server node and to deliver such an indication to the client or server software so they can clean up state. Note that the NetBIOS session service makes use of the IEEE 802.2 LLC connection-oriented services (Type 2).

IBM Corporation first introduced the NetBIOS Extended User Interface (NetBEUI) Protocol specification in 1985, as specified in [NETBEUI]. The IBM NetBEUI 3.0 transport protocol is sometimes referred to as NetBIOS frames (NBF) in Microsoft documentation (for more information, see [NBF]). NetBEUI (as specified in [NETBEUI]) implements the NetBIOS protocol and does not require a NetBIOS compatibility layer. For more information about NetBIOS, see [XOPEN-SMB] Appendix E, [NBF], and [RYAN].

When using the NetBIOS Extended User Interface as the SMB transport, the implementer MUST prepend a NetBIOS session packet header to each SMB message. The NetBIOS session packet header is defined by IBM, as specified in [NETBEUI]. The body of the SMB packet follows as a variable length payload.



Length (2 bytes): A 16-bit unsigned number in little-endian byte order that MUST be set to the length, in bytes, of the NetBIOS header (including the **Length** field). For NetBIOS session packets, this field MUST be set to 0x000E (14) to indicate there are 14 bytes in the NetBIOS session packet header.

Delimiter (2 bytes): A 16-bit field in little-endian byte order that MUST contain a value that indicates the destination for subsequent data. For NetBIOS session packets, this field MUST be set to 0xEFFF to indicate that subsequent data is destined for NetBIOS.

Command (1 byte): An 8-bit field that specifies the protocol command contained in the NetBIOS packet. This field indicates the type and function of the NetBIOS packet. For NetBIOS session packets, this field MUST be set to a command in the range from 0x14 (18) to 0x1F (31). For NetBIOS session packets, the possible values for this field are as specified in [NETBEUI] section 5.5.4.2, Table 5.10.

Data1 (1 byte): An 8-bit field of optional data that varies based on the value of the **Command** field.

Data2 (2 bytes): A 16-bit field in little-endian byte order of optional data that varies based on the **Command** field.

Transmit Correlator (2 bytes): A 16-bit field in little-endian byte order that MUST contain the correlator that is returned in a response to a given query (the value was received as the Response Correlator). This field MUST be used to correlate received response frames with transmitted requests.<4>

Response Correlator (2 bytes): A 16-bit field in little-endian byte order that MUST contain the correlator that is generated by the sender. This field MUST be used to correlate the response frame with this frame. This field is the value expected in the **Transmit Correlator** field when the response to that message is received.<5>

An 8-bit unsigned number containing the destination session number assigned at the remote node during session initialization to identify this NetBIOS session. The values of 0x00 and 0xff MUST NOT be used.

Source Number (1 byte): An 8-bit unsigned number that MUST contain the local source session number assigned during session initialization to identify this NetBIOS session. The values of 0x00 and 0xFF MUST NOT be used.

SMB Message (variable):

The body of the SMB packet.<6>

2.2 Message Syntax

A client exchanges messages with a server to access resources on the server. These messages are called Server Message Blocks (SMBs), and every SMB message has a common format.

An SMB message is the payload packet encapsulated in an SMB transport packet.

All SMB messages MUST begin with a fixed-length SMB header (as specified in section 2.2.1). The header contains a command field that indicates the operation code that the client requests or to which the server is responds. An SMB message is of variable length. The actual length depends on the SMB command field (and consequent appended data structures) and whether the SMB message is a client request or a server response.

Unless otherwise indicated, numeric fields are integers of the specified byte length.

Unless otherwise specified, multiple-byte fields (that is, 16-bit, 32-bit, and 64-bit fields) in an SMB message MUST be transmitted in little-endian byte order (least significant byte first).

Note that the [CIFS] specification uses a convention of bit N being the Nth bit from the right on a little-endian system. For consistency with other protocol documents, this document uses the standard network byte order to represent multibyte values and bit fields; and in the description of a bit specified in [CIFS], the bit N label appears as it appears in [CIFS]. The following diagram shows the bit number mappings for a 32-bit field as specified in this document to the bit numbers in [CIFS].

For example bit 14 in this document translates to bit 9 (or equivalently 0x200) in the CIFS document.

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

Unless otherwise noted, fields marked as Unused SHOULD be set to 0 when being sent, and MUST<7> be ignored when received.

[CIFS] section 2.4.2 specifies "When an error is returned, the server may choose to return only the header portion of the response SMB." For clients and servers that implement the extensions specified in this document, this is to be read as, "When an error occurs, the server SHOULD choose to return only the header portion of the response SMB, unless otherwise noted in this specification." For individual requests, the successful response is defined as follows (because the failure cases return only the header) unless otherwise noted.<8>

The SMB transport MAY fragment an SMB message into a number of smaller network transport packets when the size of the SMB message combined with any required network transport headers is larger than the size of the maximum network transport packet. For example, the **MaxBufferSize** field negotiated in the SMB_COM_NEGOTIATE server response (section 2.2.3) and the SMB_COM_SESSION_SETUP_ANDX client request (section 2.2.4) is used by the SMB client and server to set the maximum length of an SMB message that is considered to begin with an SMB header. The size of the SMB message does not include any headers added by the network transport and hardware layers to the network packet.

An Ethernet packet in compliance with version 2.0 of the Digital/Intel/Xerox (DIX) Ethernet specification can have a **Maximum Transmission Unit (MTU)** size of up to 1,500 bytes (for more information, see [RFC894]). Some IPX routers limit the MTU size of a network packet to 576 bytes. So when an SMB message of 4,096 bytes is sent by using NetBIOS over TCP, NetBIOS over IPX, direct TCP, or NetBEUI over Ethernet as the SMB transport, the SMB message is fragmented into several network transport packets by the SMB transport. All fragmentation is handled by the SMB transport at the network transport layer, not by the SMB client or server. When the fragmented packets are received, the SMB transport where the packets are received reassembles the fragments into a complete SMB message. This message is returned to the SMB client or server. All fragmentation and reassembly is handled by the SMB transport, not by SMB clients or servers that implement the SMB protocol extensions. One outcome of this fragmentation and reassembly is that larger SMB messages are fragmented by the SMB transport into several network packet fragments, with the first network packet fragment containing the SMB header and the first part of the SMB message.

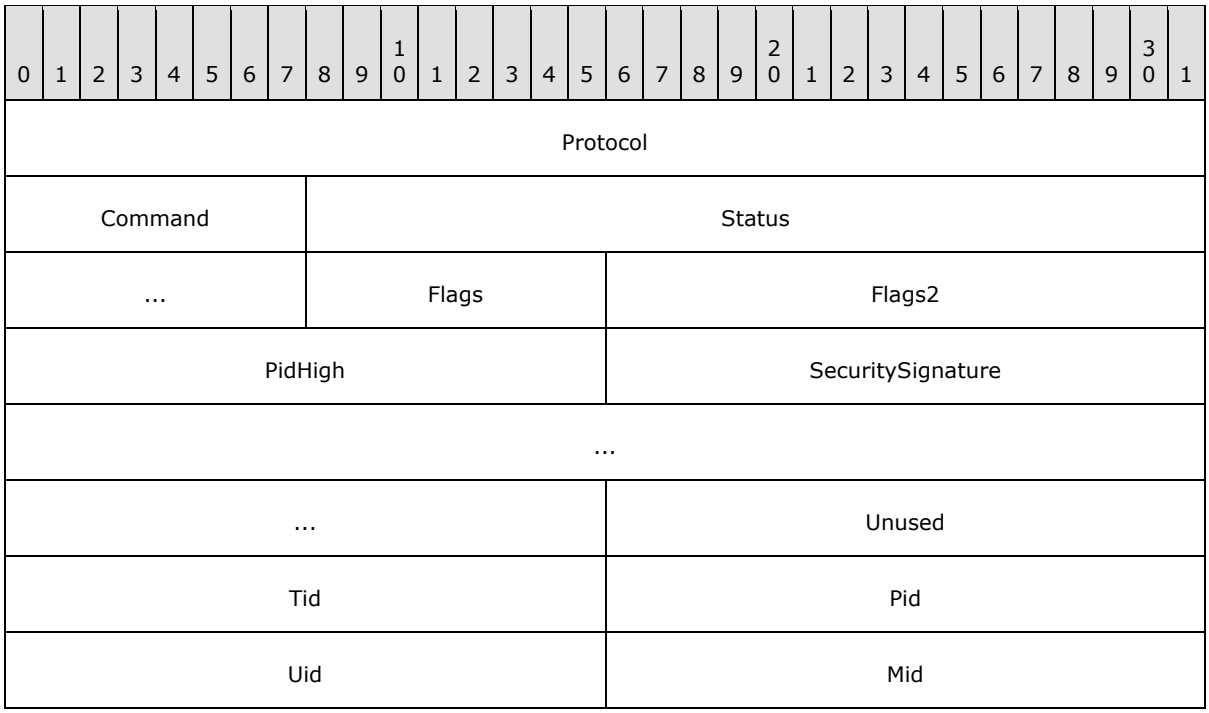
2.2.1 SMB Header Extensions and Changes

All client requests and server responses MUST begin with a fixed-size SMB header, as specified in [CIFS] section 2.4.2.

There are several new Flags2 values in the SMB header that are not in CIFS, but part of these extensions.

When message signing is active for a connection, the 8 bytes of the SMB header (the SMB header structure) previously marked as Unused and Unused2, as specified in [CIFS] section 2.4.2, MUST contain the 8-byte message signature. These fields are now called the **SecuritySignature** field. If message signing is negotiated but an SMB message is received with an invalid **SecuritySignature** field, the underlying transport connection SHOULD be immediately disconnected. For details on message signing, see section 3.1.5.1.

The following packets show the SMB header for reference.



Protocol (4 bytes): This field MUST be the 4-byte header (0xFF, S, M, B) with the letters represented by their **ASCII** (American Standard Code for Information Interchange) characters in network byte order. Note that all other multibyte fields in the SMB header MUST be encoded in little-endian byte order. This field is as specified in [CIFS] section 2.4.2.

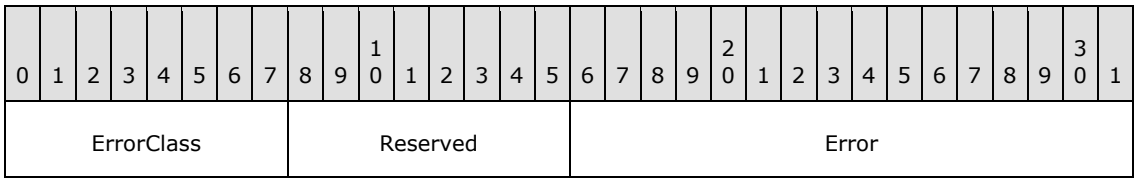
Command (1 byte): The operation code that this SMB is requesting or responding to. A list of the possible values is specified in [CIFS] section 6.1. This field is as specified in [CIFS] section 2.4.2. The following commands, listed below for completeness, are not SMB commands. The commands are used by an unrelated protocol that has adopted the SMB message headers, called the SMB Message Delivery Protocol, as specified in [MS-MSRP] section 2.2.4. An SMB client implementation MUST NOT send these commands, and an SMB server implementation MUST reject these as bad commands.

Name	Value
SMB_COM_SEND_MESSAGE	0xD0
SMB_COM_SEND_START_MB_MESSAGE	0xD5
SMB_COM_SEND_END_MB_MESSAGE	0xD6
SMB_COM_SEND_TEXT_MB_MESSAGE	0xD7

Status (4 bytes): An error code that indicates success, warning, or failure returned by the server for an operation. This field MUST indicate the error code of any failed server operation. When an error is returned, the server MAY choose to return only the header portion of the response SMB.

This field is represented in network byte order below and is as specified in [CIFS] section 2.4.2.

If a client is capable of receiving 32-bit error returns (for details on CAP_STATUS32, see sections 2.2.3 and 2.2.4), the status MUST be returned in a 32-bit error code in this field. These 32-bit error codes are specified in [MS-ERREF] section 4. Otherwise, this field MUST be interpreted as two subfields that are set by the server, and then combined to give the error code of any failed server operation, as shown below.



ErrorClass: An 8-bit value that MUST be set to the error class for this error. This field is called **Status.DosError.ErrorClass**, as specified in [CIFS]. The possible values for this field are as specified in [CIFS] section 7.

Reserved: An unused 8-bit field. This field SHOULD be set to 0 when the SMB header is sent, and MUST be ignored when the SMB header is received.

Error: A 16-bit value that MUST be set to the error code for this error. This field is called **Status.DosError.Error**, as specified in [CIFS]. The possible values for this field are as specified in [CIFS] section 7.

Flags (1 byte): The **Flags** field contains individual flags, as specified in [CIFS] sections 2.4.2 and 3.1.1. The extensions in this document do not change the use of this field.<9>

Flags2 (2 bytes): The **Flags2** field contains individual bit flags that, depending on the negotiated SMB dialect, indicate various client and server capabilities. This field is as specified in [CIFS] sections 2.4.2 and 3.1.2. Unused bit fields SHOULD be set to 0 by the server when sending a response, and MUST be ignored when received by the client. This field MAY be constructed using the following possible values.

Value	Meaning
SMB_FLAGS2_KNOWS_LONG_NAMES 0x0001	If set in a client request for directory enumeration, the server MAY return long names (that is, names that are not **8.3 names**) in the response to this request. If not set in a client request for directory enumeration, the server MUST only return 8.3 names in the response to this request. This flag indicates that in a direct enumeration request, paths are not restricted to 8.3 names by the server. This bit field SHOULD be supported when LM1.2X002 or later is negotiated for the SMB dialect. This bit field is called bit 0, as specified in [CIFS] section 3.1.2.
SMB_FLAGS2_KNOWS_EAS 0x0002	If set in a client request, the client is aware of extended attributes. The client MUST set this bit if the client is aware of extended attributes. In response to a client request with this flag set, a server MAY include extended attributes in the response. This bit field SHOULD be supported when LM1.2X002 or later is

Value	Meaning
	negotiated for the SMB dialect. This bit field is called bit 1, as specified in [CIFS] section 3.1.2.
SMB_FLAGS2_SMB_SECURITY_SIGNATURE 0x0004	If set, the client is requesting signing (if not active) or the message being sent is signed, as specified in section 3.1.4.1. This bit is used on the SMB header of an SMB_COM_SESSION_SETUP_ANDX client request (section 2.2.4) to indicate that the client supports signing and the server can choose to enforce signing on the connection based on its configuration. If the server wants to turn on signing for this connection, it MUST set this flag and also sign the SMB_COM_SESSION_SETUP_ANDX response (section 2.2.5)—after which all traffic on the connection MUST be signed. On the SMB header of other SMB client requests, the setting of this bit indicates that the packet has been signed. This bit field SHOULD be supported when NTLM 0.12 or later is negotiated for the SMB dialect.
SMB_FLAGS2_COMPRESSED 0x0008	If set by the client, the client is requesting compressed data for an SMB_COM_READ_ANDX request. If cleared by the server, the server is notifying the client that the data was written uncompressed.
SMB_FLAGS2_SMB_SECURITY_SIGNATURE_REQUIRED 0x0010	This flag MUST be set by the client on the first SMB_COM_SESSION_SETUP_ANDX request (section 2.2.4) sent to a server that supports extended security, if the client requires all further communication with this server to be signed. If the server does not support signing, it MUST disconnect the client by closing the underlying transport connection. Clients and servers MUST ignore this value for all other requests/responses. If the client receives a non-signed response from the server, it MUST disconnect the underlying transport connection immediately. This bit field SHOULD be supported when NTLM 0.12 or later is negotiated for the SMB dialect.<10>
SMB_FLAGS2_IS_LONG_NAME 0x0040	If set, the path contained in the message contains long names; otherwise, the paths are restricted to 8.3 names. This bit field SHOULD be supported when NTLM 0.12 or later is negotiated for the SMB dialect.

Value	Meaning
SMB_FLAGS2_REPARSE_PATH 0x0400	If set, the path in the request MUST contain an **@GMT token** (that is, a previous version token), as specified in section 3.2.4.3.1. This bit field SHOULD be supported when NTLM 0.12 or later is negotiated for the SMB dialect.<11>
SMB_FLAGS2_EXTENDED_SECURITY 0x0800	Indicates that the client supports SPNEGO authentication, as specified in section 3.2.4.2.3. This bit field SHOULD be supported when NTLM 0.12 or later is negotiated for the SMB dialect.
SMB_FLAGS2_DFS 0x1000	This bit field is called bit 12, as specified in [CIFS] section 3.1.2. The client MUST set this field for all requests with a valid TID if all of the following conditions are met. 1. The server supports DFS (as indicated by the CAP_DFS flag specified in section 2.2.3). 2. NTLM 0.12 or later is negotiated for the SMB dialect. 3. The share is a DFS share. (as indicated by the SMB_SHARE_IS_IN_DFS flag specified in section 2.2.7). 4. The operation is targeted at a DFS namespace as indicated by the application via the higher level action specified in section 3.2.4.2). For SMB_COM_TREE_CONNECT_ANDX requests (section 2.2.6), only conditions (1), (2) and (4) above MUST be satisfied. For all other requests, the client MUST set this field to 0 and the server MUST ignore this field.
SMB_FLAGS2_PAGING 0x2000	If set in a client request, a read MAY be permitted if the client does not have read permission but does have execute permission. This flag is only useful on a read request. This bit field SHOULD be supported when LM1.2X002 or later is negotiated for the SMB dialect. This bit field is called bit 13, as specified in [CIFS]

Value	Meaning
	section 3.1.2.<12>
SMB_FLAGS2_NT_STATUS 0x4000	If set in a server response, the returned error code MUST be a 32-bit error code in **Status.Status** field. Otherwise, the **Status.DosError.ErrorClass** and **Status.DosError.Error** fields MUST contain the MS-DOS-style error information. When passing Windows NT status codes is negotiated (see SMB_COM_NEGOTIATE Server Response Extension and SMB_COM_SESSION_SETUP_ANDX Client Request Extension), this flag MUST be set for every SMB. This bit field SHOULD be supported when NTLM 0.12 or later is negotiated for the SMB dialect. This bit field is called bit 14, as specified in [CIFS] section 3.1.2.
SMB_FLAGS2_UNICODE 0x8000	If set in a client request or server response, any fields that contain strings in this SMB message MUST be encoded as an array of 16-bit **Unicode** characters. Otherwise, these fields MUST be encoded as an array of **OEM characters**. This bit field SHOULD be supported when NTLM 0.12 or later is negotiated for the SMB dialect. This bit field is called bit 15, as specified in [CIFS] section 3.1.2.

PidHigh (2 bytes): This field MUST give the 2 high bytes of the **process identifier** if the client wants to use 32-bit process IDs, as specified in [CIFS] section 2.4.2. If a client uses 16-bit process IDs, this field MUST be set to 0.<13>

SecuritySignature (8 bytes): This field is the combination of the fields defined as Unused and Unused2, as specified in [CIFS] section 2.4.2. It SHOULD be used to carry a secure signature when message signing is enabled, as specified in section 3.1.4.1.

Unused (2 bytes): This field is unused. The sender SHOULD set this to 0, and the receiver MUST ignore this field.

Tid (2 bytes): This field identifies the subdirectory (or tree) (also referred to as a share in this document) on the server that the client is accessing, as specified in [CIFS] sections 2.4.2 and 3.1.3. The description of this field, as specified in [CIFS] section 2.4.2, SHOULD be corrected as follows: For messages that do not reference a particular tree, this field MUST be set to 0xFFFF or 0. Both values are acceptable.

Pid (2 bytes): Caller's process ID, as specified in [CIFS] sections 2.4.2 and 3.1.4; it MUST be generated by the client to uniquely identify a process within the client computer. If the client wants to use 32-bit process IDs, it also MUST set the **PidHigh** field to the two most significant bytes of the caller's process ID.

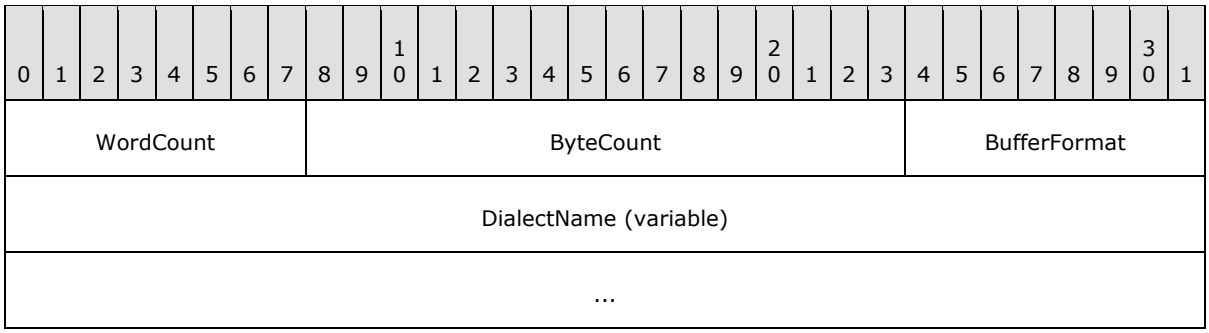
Uid (2 bytes): This field SHOULD identify the authenticated instance of the implementer, as specified in [CIFS] sections 2.4.2 and 2.8.

Mid (2 bytes): This field SHOULD be the multiplex ID that is used to associate a response with a request, as specified in [CIFS] sections 2.4.2 and 3.1.5.

2.2.2 SMB_COM_NEGOTIATE Client Request Extension

An SMB_COM_NEGOTIATE request MUST be sent by a client as the initial packet that handles dialect and capability negotiation. The client SHOULD send a list of SMB dialects with which it can communicate. The server response MUST be a selection of one of those dialects or an error indicating that none of the dialects were acceptable. The SMB_COM_NEGOTIATE message MUST be sent from the client to the server to establish an SMB connection. Only one SMB_COM_NEGOTIATE request SHOULD be sent between a client and a server over an SMB transport while an SMB connection is active on the SMB transport. Subsequent SMB_COM_NEGOTIATE requests from a client to a server over the same SMB transport while an SMB connection is still established SHOULD be rejected with an error response, and no action SHOULD be taken.<14>

The client request for negotiation is identical in this specification to what is specified in the base protocol, with the exception of the new flags in the SMB Header, as specified in section 2.2.1. The SMB_FLAGS2_EXTENDED_SECURITY bit, when set, indicates support for specification [RFC4178] and GSS authentication. For more information, see section 3.1.5.1.



WordCount (1 byte): Word count for this request MUST be 0 because there are zero 16-bit WORDs between the **WordCount** field and the **ByteCount** field.

ByteCount (2 bytes): Count of data bytes in the packet. This field has a minimum value of 0x0002. This field MUST be the total length of the combined **BufferFormat** and **DialectName** fields.

BufferFormat (1 byte): Data format of the buffer being sent in the **DialectName** field. This field MUST be set to 0x02 to indicate that the **DialectName** field is an array of null-terminated ASCII strings, separated by the character with ASCII code 0x2.

DialectName (variable): Array of null-terminated ASCII strings that indicate the SMB dialects supported by the client. The protocol does not impose any particular structure on the SMB dialect strings. Implementers of particular protocols MAY choose to include, for example, version numbers in the string.

To support the extended version of the SMB protocol, the client MUST include the NTLM 0.12 dialect in the SMB_COM_NEGOTIATE client request.<15>

2.2.3 SMB_COM_NEGOTIATE Server Response Extension (Extended Security)

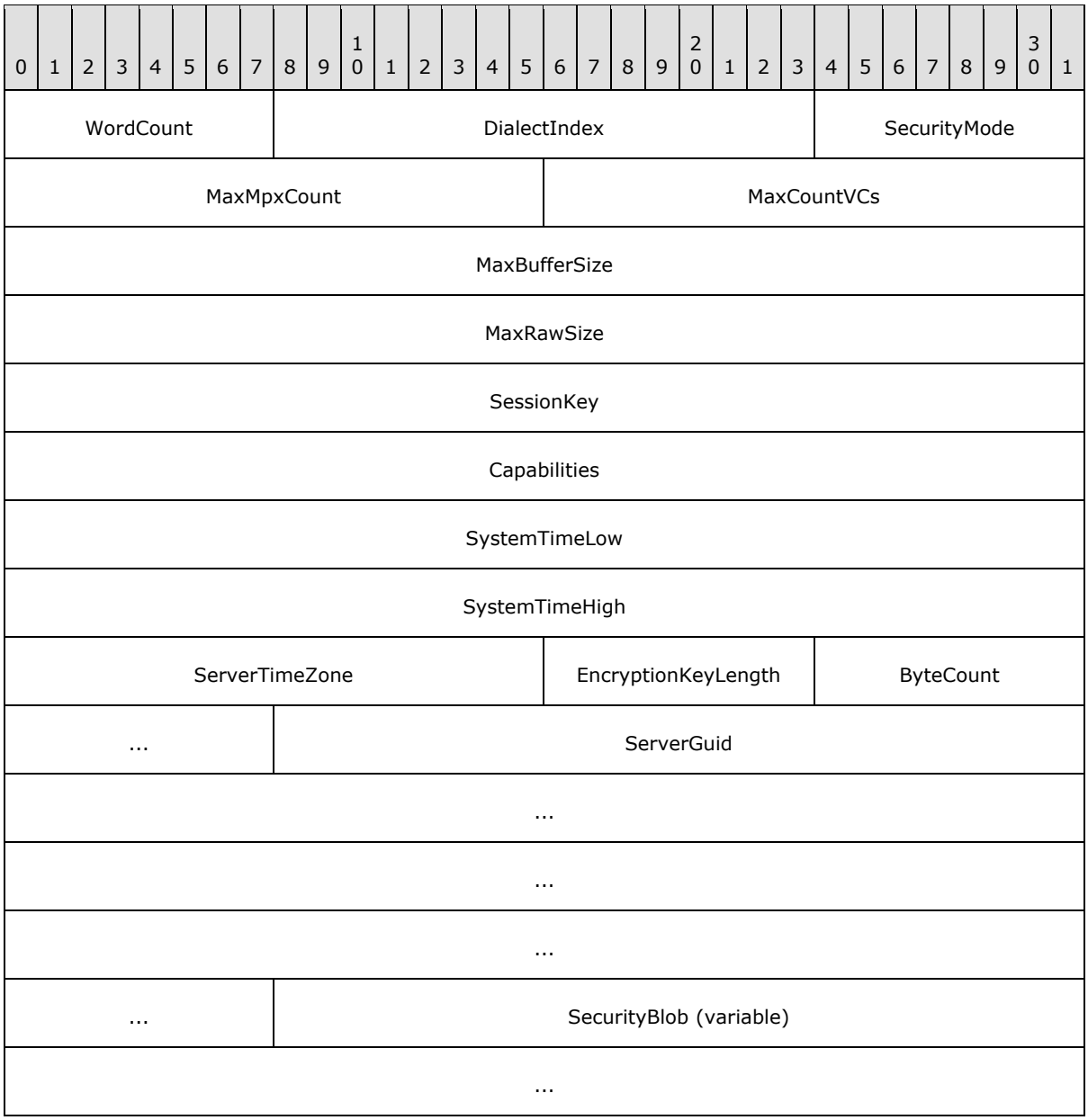
An SMB_COM_NEGOTIATE response MUST be sent by a server in reply to a SMB_COM_NEGOTIATE client request when the request is successful.

The server response MUST include a selection of one of the SMB dialects passed in the **DialectName** field of the SMB_COM_NEGOTIATE client request, or an error if none of the SMB dialects are supported. The server MUST return the selected SMB dialect as a number (starting from 0) that represents the array index in the **DialectName** field of the SMB_COM_NEGOTIATE client request. The selected SMB dialect number MUST be returned in the **DialectName** field of the server response. If none of the dialects are acceptable, the server MUST return 0xffff in the **DialectName** field, the **WordCount** MUST be set to 1 and the **Status** field in the SMB Header MUST be set to STATUS_NOT_SUPPORTED in the response.

Only one SMB_COM_NEGOTIATE request SHOULD be sent between a client and a server over an SMB transport while an SMB connection is active on the SMB transport. Subsequent SMB_COM_NEGOTIATE requests from a client to a server over the same SMB transport while an SMB connection is still established SHOULD be rejected by the server with an STATUS_INVALID_SMB error response (as specified in section 2.2.17) and no action SHOULD be taken.<16>

If none of the requested SMB dialects are acceptable, the server MUST set the **DialectIndex** field to 0xffff in the SMB_COM_NEGOTIATE response structure (as specified in [CIFS] section 4.1.1), and MUST indicate in the SMB Header response that an error has occurred.

If extended security is being used and one of the dialects IS acceptable, the response MUST take the following form.



WordCount (1 byte): Word count for this response MUST be 0x11 (17) because there are seventeen 16-bit WORDs between the **WordCount** field and the **ByteCount** field.

DialectIndex (2 bytes): Array index of the SMB dialect that was selected from the **DialectName** field that was passed in the SMB_COM_NEGOTIATE client request. This field MUST be a number (starting from 0) that represents the array index in the **DialectName** field. To support the extended version of the SMB Protocol, the server MUST select the array index of the NTLANMAN dialect (for more information, see SMB_COM_NEGOTIATE Client Request Extension).

SecurityMode (1 byte): Security mode field MUST be structured as shown below. How to convert these to bit positions is specified in [CIFS] section 2.2. Unused bit fields SHOULD be

set to 0 by the server when sending a response, and MUST be ignored when received by the client.

Value	Meaning
NEGOTIATE_USER_SECURITY 0x01	When set, indicates user mode security. When not set, indicates share mode security, as specified in [CIFS] section 4.1.1 Bit 0).
NEGOTIATE_ENCRYPT_PASSWORDS 0x02	When set, indicates the client MUST encrypt passwords. When not set, indicates the client MUST send plain-text passwords for user level security mode and share level security mode, as specified in section [CIFS] section 4.1.1, "Bit 1".
NEGOTIATE_SECURITY_SIGNATURES_ENABLED 0x04	When set, security signatures (SMB sequence numbers) MAY be used.
NEGOTIATE_SECURITY_SIGNATURES_REQUIRED 0x08	When set, security signatures (SMB sequence numbers) MUST be used.

MaxMpxCount (2 bytes): Maximum pending **multiplexed requests** supported by the server.

MaxCountVCs (2 bytes): Maximum **virtual connections (VCs)** between the client and the server.<17>

MaxBufferSize (4 bytes): Maximum size, in bytes, of the server buffer for sending and receiving SMB messages. This is the size of the largest message that the client MAY send to the server. This is the size of the buffer used for the SMB message from the start of the SMB Header to the end of the packet. This size is not the size of the complete network packet because it does not include the size that is needed for the underlying transport and the SMB transport header. The only exceptions when this buffer size MAY be exceeded are the SMB_COM_READ_ANDX command if the client and server both support the CAP_LARGE_READX capability, and the SMB_COM_WRITE_ANDX command if the client and server both support CAP_LARGE_WRITEX capability (see the **Capabilities** field below).<18>

MaxRawSize (4 bytes): Maximum raw buffer size, in bytes. This specifies the maximum message size the server MAY receive in an SMB_COM_WRITE_RAW client request, and the maximum message size the server MAY send in an SMB_COM_READ_RAW server response.<19>

SessionKey (4 bytes): A unique session identifier that the server MUST generate to identify this session. The client uses this value in subsequent SMB client requests for this session. The server generates the session key value and the client MUST send this value on subsequent SMB client requests for this session. The server can choose any value. The client always echoes back the server value in its response.

Capabilities (4 bytes): A set of server capabilities. This bit field SHOULD be used by the server to notify the client as to which features are supported by the server. There are several new capability bits: CAP_COMPRESSED_DATA, CAP_DYNAMIC_REAUTH, CAP_EXTENDED_SECURITY, CAP_INFOLEVEL_PASSTHRU, CAP_LARGE_READX, CAP_LARGE_WRITEX, CAP_LWIO, and CAP_UNIX. Any value not listed in the table below SHOULD be unused. A server SHOULD set the unused bits to 0 in a response, and a client MUST ignore these bits.

Value	Meaning
CAP_RAW_MODE 0x00000001	The server supports SMB_COM_READ_RAW and SMB_COM_WRITE_RAW requests.
CAP_MPX_MODE 0x00000002	The server supports SMB_COM_READ_MPX and SMB_COM_WRITE_MPX requests.
CAP_UNICODE 0x00000004	The server supports 16-bit Unicode characters.
CAP_LARGE_FILES 0x00000008	The server supports large files with 64-bit offsets.
CAP_NT_SMBS 0x00000010	The server supports the SMB packets particular to the NTLM 0.12 dialect.
CAP_RPC_REMOTE_APIS 0x00000020	The server supports the use of Microsoft RPC for remote API calls that would otherwise use the legacy Remote Administration Protocol, as specified in [MS-RAP].
CAP_STATUS32 0x00000040	The server is capable of responding with 32-bit status codes in the **Status** field of the SMB Header (for more information, see 2.2.1).
CAP_LEVEL_II_OPLOCKS 0x00000080	The server supports level II **oplocks**.
CAP_LOCK_AND_READ 0x00000100	The server supports the SMB_COM_LOCK_AND_READ command requests.
CAP_NT_FIND 0x00000200	The server supports the TRANS2_FIND_FIRST2, TRANS2_FIND_NEXT2, and FIND_CLOSE2 command requests.
CAP_DFS 0x00001000	The server is aware of the Distributed File System (DFS) Referral Protocol, as specified in [MS-DFSC], and can respond to Microsoft DFS referral requests. For more information, see [MSDFS].
CAP_INFOLEVEL_PASSTHRU 0x00002000	The server supports Windows NT information level requests, as specified in section 2.2.13. This allows the client to pass native Windows NT structures (as specified in [MS-FSCC]) in QUERY and SET operations, as specified in section 2.2.13.
CAP_LARGE_READX 0x00004000	The server supports large read operations. This capability affects the maximum size, in bytes, of the server buffer for sending an SMB_COM_WRITE_ANDX response to the client. When this capability is set by the server (and set by the client in the SMB_COM_SESSION_SETUP_ANDX request (section 2.2.4)), the maximum server buffer size for sending data can be up to 65,535 bytes rather than the **MaxBufferSize** field. Therefore, the server can send a single SMB_COM_WRITE_ANDX response to the client up to this size.
CAP_LARGE_WRITEX 0x00008000	The server supports large write operations. This capability affects the maximum size, in bytes, of the server buffer for receiving an SMB_COM_WRITE_ANDX client request. When this capability is set by the server (and set by the client in the SMB_COM_SESSION_SETUP_ANDX request (section 2.2.4)), the maximum server buffer size can be up to 65,535 bytes rather than the **MaxBufferSize** field. Therefore, a client can send a single SMB_COM_WRITE_ANDX request up to this size.

Value	Meaning
CAP_LWIO 0x00010000	The server supports the FSCTL_SRV_REQUEST_RESUME_KEY (as specified in sections 2.2.14.7.2 and 2.2.14.8.2) FSCTL that was sent as an SMB_COM_NT_TRANSACTION_REQUEST with an NT_TRANSACT_IOCTL subcommand, as specified in section 2.2.14.72.2.14.7.
CAP_UNIX 0x00800000	The server supports UNIX extensions. For more information, see [SNIA].
CAP_COMPRESSED_DATA 0x02000000	The server supports compressed SMB packets.<20>
CAP_DYNAMIC_REAUTH 0x20000000	The server supports reauthentication, if required, as specified in section 3.1.5.1.
CAP_PERSISTENT_HANDLES 0x40000000	The server supports persistent handles.
CAP_EXTENDED_SECURITY 0x80000000	The server supports extended security for authentication, as specified in section 3.2.4.2.3. This bit is used in conjunction with the SMB_FLAGS2_EXTENDED_SECURITY Flags2 field in the SMB Header (bit 11), as specified in SMB Header Extensions and Changes.

SystemTimeLow (4 bytes): The system **UTC (Coordinated Universal Time)** of the server (least significant four bytes). The combined 8-byte value of **SystemTimeLow** and **SystemTimeHigh** MUST be in the TIME format, as specified in [CIFS] section 3.5.

SystemTimeHigh (4 bytes): The system UTC of the server (most significant four bytes). The combined eight-byte value of SystemTimeLow and SystemTimeHigh MUST be in the TIME format, as specified in [CIFS] section 3.5.

ServerTimeZone (2 bytes): The time zone of the server MUST be expressed in minutes from UTC. This field is a signed 16-bit integer. This field represents minutes measured plus or minus from UTC for the time zone of the server. Note that [CIFS] section 4.1.1 specified this field incorrectly as a **USHORT**.

EncryptionKeyLength (1 byte): If the CAP_EXTENDED_SECURITY bit is set, the server MUST set this value to 0 and clients MUST ignore this value. When extended security is not used, the SMB_COM_NEGOTIATE server response (section 2.2.2) is as specified below and in [CIFS] section 4.1.1.

ByteCount (2 bytes): The value MUST be the number of data bytes in the packet.

ServerGuid (16 bytes): The **globally unique identifier (GUID)** generated by the server to uniquely identify this server. This field SHOULD NOT be used by a client as a secure method of identifying a server because it can easily be faked. A client MAY use this information to detect if connections to different textual names resolve to the same target server when TCP port 445 is used as a transport. This knowledge can then be used to appropriately set the **VcNumber** field in the SMB_COM_SESSION_SETUP_ANDX request (section 2.2.4), as specified in [CIFS] section 4.1.2.<21>

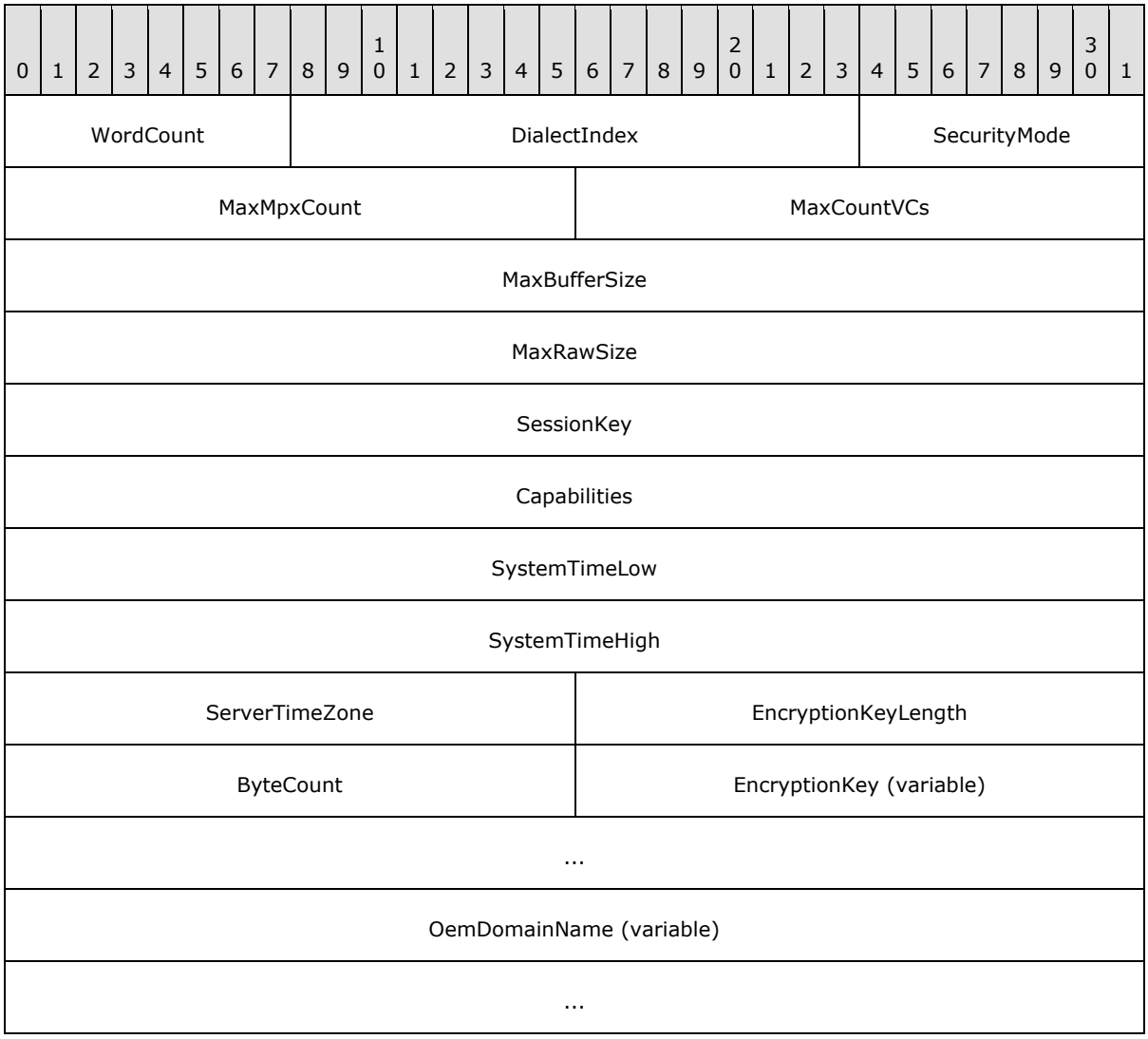
SecurityBlob (variable): A security binary large object (BLOB) that MUST contain an authentication token as produced by the GSS protocol (as specified in section 3.2.4.2.3 and [RFC4178]).

See Security Blob (section 2.2.3.1) for the specification of this field.

2.2.3.1 SMB_COM_NEGOTIATE Server Response Extension (Non-Extended Security)

The SecurityBlob packet MUST contain an authentication token as produced by the GSS protocol (as specified in section 3.2.4.2.3 and [RFC4178]).

If extended security is not being used and one of the dialects is acceptable, the response MUST take the following form.



WordCount (1 byte): The word count for this response MUST be 0x11 (17) because there are seventeen 16-bit WORDs between the **WordCount** field and the **ByteCount** field.

DialectIndex (2 bytes): An array index of the SMB dialect that was selected from the **DialectName** field that was passed in the SMB_COM_NEGOTIATE client request (section 2.2.2). This field MUST be a number (starting from 0) that represents the array index in the

DialectName field. To support the extended version of the SMB Protocol, the server MUST select the array index of the NTLANMAN dialect. For more details, see section 2.2.2.

SecurityMode (1 byte): The **SecurityMode** field MUST be structured, as shown below. Note that the CIFS Protocol (as specified [CIFS]) numbers the bits in the opposite direction from conventional protocol specification use. Unused bit fields SHOULD be set to 0 by the server when sending a response, and MUST be ignored when received by the client.

Value	Meaning
NEGOTIATE_USER_SECURITY 0x00	When set, indicates user mode security. When not set, indicates share mode security, as specified in [CIFS] 4.1.1, "Bit 0".
NEGOTIATE_ENCRYPT_PASSWORDS 0x01	When set, indicates to encrypt passwords, as specified in [CIFS] 4.1.1, "Bit 1".

MaxMpxCount (2 bytes): The maximum pending multiplexed requests supported by the server.

MaxCountVCs (2 bytes): The maximum virtual connections (VCs) between the client and the server.<22>

MaxBufferSize (4 bytes): The maximum size, in bytes, of the server buffer for sending and receiving SMB messages. This is the size of the largest message the client MAY send to the server. This is the size of the buffer used for the SMB message from the start of the SMB header to the end of the packet. This size is not the size of the complete network packet because it does not include the size needed for the underlying transport and the SMB transport header. The only exceptions when this buffer size can be exceeded are the SMB_COM_READ_ANDX command if the client and server both support the CAP_LARGE_READX capability, and the SMB_COM_WRITE_ANDX command if the client and server both support the CAP_LARGE_WRITEX capability (see the **Capabilities** field below).<23>

MaxRawSize (4 bytes): The maximum raw buffer size, in bytes. This specifies the maximum message size the client MAY send in an SMB_COM_WRITE_RAW client request, and the maximum message size the client MAY receive from the server in an SMB_COM_READ_RAW server response.<24>

SessionKey (4 bytes): The server MUST set this value to 0, and the client MUST ignore this value.

Capabilities (4 bytes): A set of server capabilities. This bit field SHOULD be used by the server to notify the client as to which features are supported by the server. There are several new capability bits: CAP_COMPRESSED_DATA, CAP_DYNAMIC_REAUTH, CAP_INFOLEVEL_PASSTHRU, CAP_LARGE_READX, CAP_LARGE_WRITEX, CAP_LWIO, and CAP_UNIX. Any value not listed in the table below is unused. A server SHOULD set the unused bits to 0 in a response, and a client SHOULD ignore these bits.

Value	Meaning
CAP_RAW_MODE 0x00000001	The server supports SMB_COM_READ_RAW and SMB_COM_WRITE_RAW requests.
CAP_MPX_MODE 0x00000002	The server supports SMB_COM_READ_MPX and SMB_COM_WRITE_MPX requests.

Value	Meaning
CAP_UNICODE 0x00000004	The server supports 16-bit Unicode characters.
CAP_LARGE_FILES 0x00000008	The server supports large files with 64-bit offsets.
CAP_NT_SMBS 0x00000010	Server supports the SMB packets particular to the NTLM 0.12 dialect.
CAP_RPC_REMOTE_APIS 0x00000020	The server supports the use of Microsoft RPC for remote API calls that would otherwise use the legacy Remote Administration Protocol, as specified in [MS-RAP].
CAP_STATUS32 0x00000040	The server is capable of responding with 32-bit status codes in the **Status** field of the SMB header, as specified in section 2.2.12.2.1.
CAP_LEVEL_II_OPLOCKS 0x00000080	The server supports level II oplocks.
CAP_LOCK_AND_READ 0x00000100	The server supports the SMB_COM_LOCK_AND_READ command requests.
CAP_NT_FIND 0x00000200	The server supports the TRANS2_FIND_FIRST2, TRANS2_FIND_NEXT2, and FIND_CLOSE2 command requests.
CAP_DFS 0x00001000	The server is aware of the Distributed File System (DFS) Referral Protocol, as specified in [MS-DFSC], and can respond to Microsoft DFS referral requests. For more information, see [MSDFS].
CAP_INFOLEVEL_PASSTHRU 0x00002000	The server supports Windows NT information level requests, as specified in section 2.2.13. This allows the client to pass native Windows NT structures in QUERY and SET operations, as specified in section 2.2.13. Native Windows NT structures are specified in [MS-FSCC].
CAP_LARGE_READX 0x00004000	The server supports large read operations. This capability affects the maximum size, in bytes, of the server buffer for sending an SMB_COM_WRITE_ANDX response to the client. When this capability is set by the server (and set by the client in the SMB_COM_SESSION_SETUP_ANDX request (section 2.2.4)), the maximum server buffer size for sending data can be up to 65,535 bytes rather than the **MaxBufferSize** field. Therefore, the server can send a single SMB_COM_WRITE_ANDX response to the client up to this size.
CAP_LARGE_WRITEX 0x00008000	The server supports large write operations. This capability affects the maximum size, in bytes, of the server buffer for receiving an SMB_COM_WRITE_ANDX client request. When this capability is set by the server (and set by the client in the SMB_COM_SESSION_SETUP_ANDX request (section 2.2.4))), the maximum server buffer size can be up to 65,535 bytes rather than the **MaxBufferSize** field. Therefore, a client can send a single SMB_COM_WRITE_ANDX request up to this size.
CAP_LWIO 0x00010000	The server supports the FSCTL_SRV_REQUEST_RESUME_KEY (as specified in sections 2.2.14.7.2 and 2.2.14.8.2) that FSCTL sent as an SMB_COM_NT_TRANSACTION_REQUEST with an NT_TRANSACT_IOCTL subcommand, as specified in section

Value	Meaning
	2.2.14.7.
CAP_UNIX 0x00800000	The server supports UNIX extensions. For more information, see [SNIA].
CAP_COMPRESSED_DATA 0x02000000	The server supports compressed SMB packets.<25>
CAP_DYNAMIC_REAUTH 0x20000000	The server supports reauthentication, if required, as specified in section 3.1.5.1.
CAP_PERSISTENT_HANDLES 0x40000000	The server supports persistent handles.

SystemTimeLow (4 bytes): The system UTC (Coordinated Universal Time) of the server (least significant four bytes). The combined eight-byte value of **SystemTimeLow** and **SystemTimeHigh** MUST be in the TIME format, as specified in [CIFS] section 3.5.

SystemTimeHigh (4 bytes): The system UTC of the server (most significant four bytes). The combined eight-byte value of **SystemTimeLow** and **SystemTimeHigh** MUST be in the TIME format, as specified in [CIFS] section 3.5.

ServerTimeZone (2 bytes): The time zone of the server MUST be expressed in minutes from UTC. This field is a signed 16-bit integer. This field represents minutes measured plus or minus from UTC for the time zone of the server. Note that [CIFS] section 4.1.1 specifies this field incorrectly as a USHORT, as specified in MS-DYTP section **2.2.57**.

EncryptionKeyLength (2 bytes): An 8-bit unsigned integer that MUST represent the length, in bytes, of the **EncryptionKey** field.

ByteCount (2 bytes): The number of data bytes in the packet.

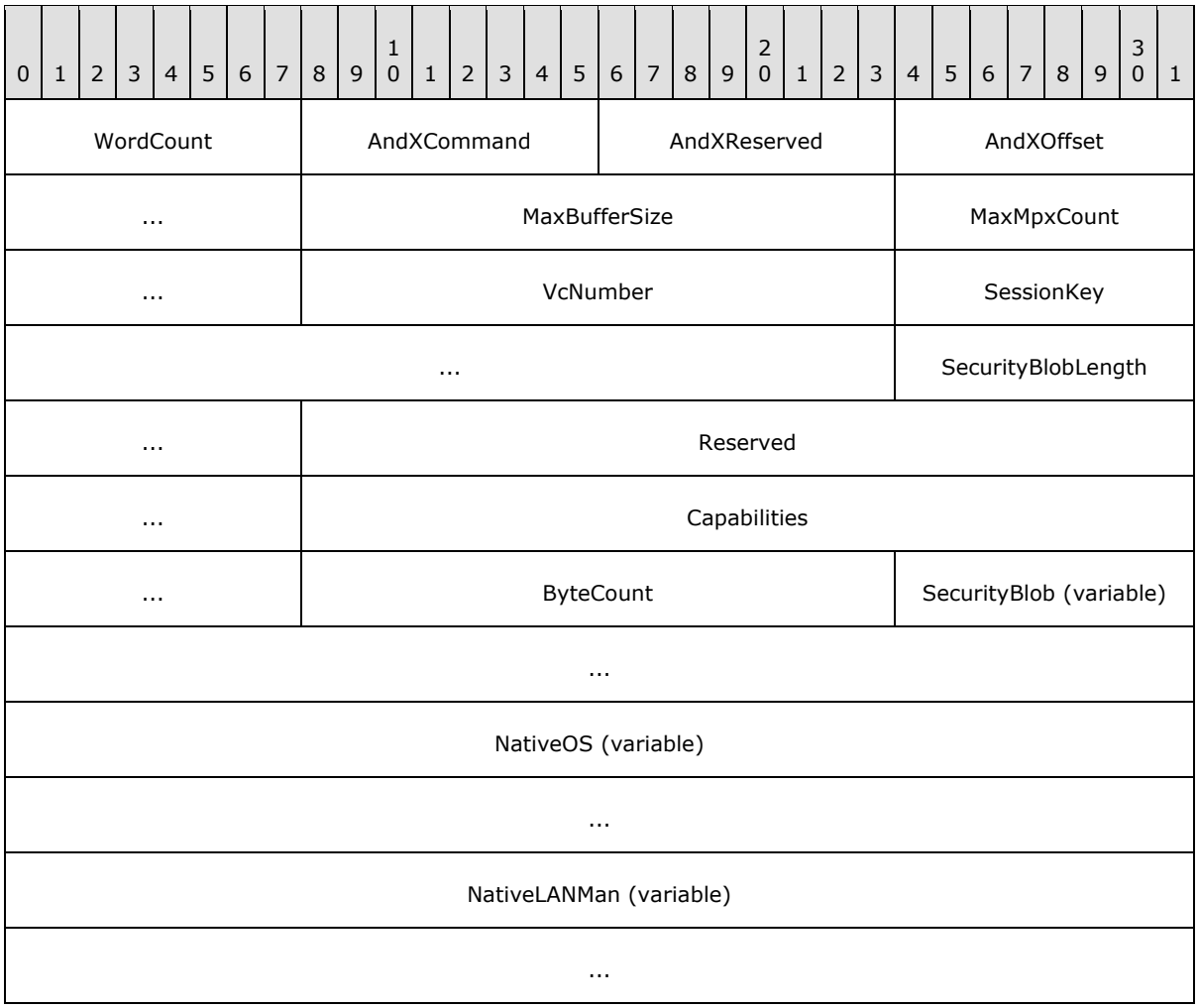
EncryptionKey (variable): An array of unsigned characters that must be **EncryptionKeyLength** bytes long, and MUST represent the challenge encryption key. This field is not null terminated.<26>

OemDomainName (variable): The name of the server's **domain** or workgroup encoded as a variable-length, null-terminated Unicode string.

2.2.4 SMB_COM_SESSION_SETUP_ANDX Client Request Extension

An SMB_COM_SESSION_SETUP_ANDX request MUST be sent by a client to begin user authentication on an SMB connection and establish an **SMB session**, as specified in [CIFS] section 4.1.2.

When extended security is being used, as specified in section 3.2.4.2.3, the request MUST take the following form instead of what is specified in [CIFS] section 4.1.2. Other than the specific differences in the **WordCount**, **Capabilities**, **SecurityBlobLength**, and **SecurityBlob** fields, all other fields are as specified in the corresponding request structure in [CIFS] section 4.1.2.



WordCount (1 byte): The word count for this request MUST be 0x0C (12) because there are twelve 16-bit WORDs between the **WordCount** and the **ByteCount** fields.

AndXCommand (1 byte): The secondary SMB command in the packet. This value MUST be set to 0xFF if there are no additional SMB commands in the client request packet. Information about compounded requests is specified in [CIFS] section 3.12.

AndXReserved (1 byte): A reserved field. This MUST be set to 0 when this request is sent, and the server MUST ignore this value when the message is received. Information about compounded requests is specified in [CIFS] section 3.12.

AndXOffset (2 bytes): This field MUST be set to the offset in bytes from the start of the SMB header to the start of the **WordCount** field in the next SMB command in this packet. The field is valid only if the **AndXCommand** field is not set to 0xFF. Information about compounded requests is specified in [CIFS] section 3.12.

MaxBufferSize (2 bytes): The maximum size, in bytes, of the client buffer for sending and receiving SMB messages. This is the size of the largest message that the server SHOULD send to the client. This is the size of the buffer used for the SMB message from the start of the SMB header to the end of the packet. This size is not the size of the complete network packet

because it does not include the size needed for the underlying transport and the SMB transport header. The only exceptions where this buffer size can be exceeded are the SMB_COM_READ_ANDX command if the client and server both support the CAP_LARGE_READX capability, and the SMB_COM_WRITE_ANDX command if the client and server both support the CAP_LARGE_WRITEX capability (see the **Capabilities** field below).<27>

MaxMpxCount (2 bytes): Maximum pending multiplexed requests supported by the client.

VcNumber (2 bytes): The number of this virtual connection between the client and the server. This field SHOULD be set to a value of 0 for the first virtual connection between the client and the server and SHOULD be set to a unique non-zero value for additional virtual connections using this SMB connection.<28>

SessionKey (4 bytes): The client MUST set this to be equal to the **SessionKey** in the SMB_COM_NEGOTIATE response (section 2.2.3).<29>

SecurityBlobLength (2 bytes): This value MUST specify the length in bytes of the variable-length **SecurityBlob** contained within the request.

Reserved (4 bytes): An unused value that SHOULD be set to 0 when sending this message. The server MUST ignore this field when receiving this message.

Capabilities (4 bytes): A set of client capabilities. These flags are a subset of those specified in section 2.2.3 for the server capabilities returned in the SMB_COM_NEGOTIATE response. The possible client capabilities MAY only include a combination of the following: CAP_DYNAMIC_REAUTH, CAP_EXTENDED_SECURITY, CAP_LEVEL_II_OPLOCKS, CAP_NT_SMBS, CAP_NT_STATUS, and CAP_UNICODE. Servers MUST NOT check for Kerberos ticket expiry if the client does not support the CAP_DYNAMIC_REAUTH capability.<30>

ByteCount (2 bytes): This field MUST be the number of data bytes in the Data buffer in this packet. If SMB_FLAGS2_UNICODE is set in the **Flags2** field of the SMB header of the request, this field has a minimum value of 4. If SMB_FLAGS2_UNICODE is not set, this field has a minimum value of 2. This field is the total length of the combined **SecurityBlob**, **NativeOS**, and **NativeLANMan** fields.

SecurityBlob (variable): This field MUST be the authentication token sent to the server, as specified in section 3.2.4.2.3 and [RFC4178].

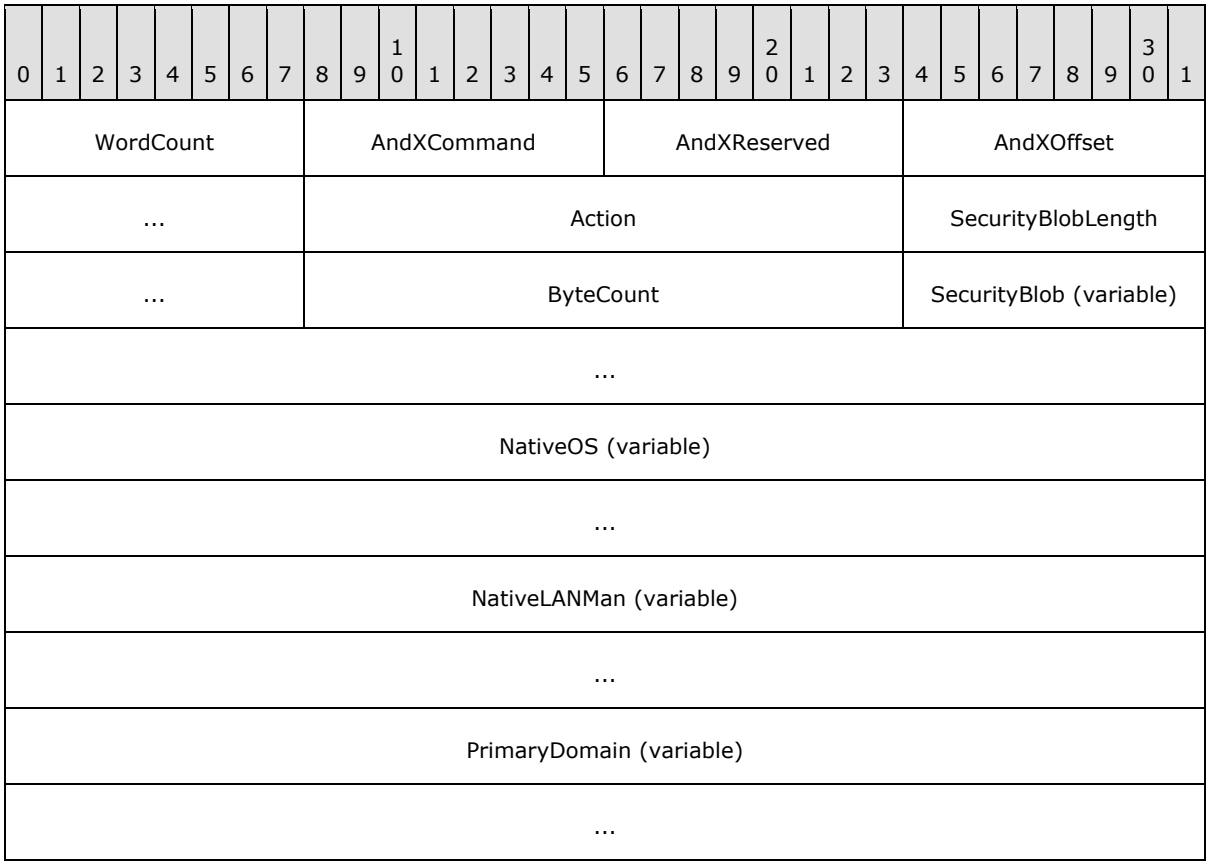
NativeOS (variable): A string that represents the native operating system of the SMB client. If SMB_FLAGS2_UNICODE is set in the **Flags2** field of the SMB header of the request, the name string MUST be a null-terminated array of 16-bit Unicode characters. Otherwise, the name string MUST be a null-terminated array of OEM characters. If the name string consists of Unicode characters, this field MUST be aligned to start on a two-byte boundary from the start of the SMB header.<31>

NativeLANMan (variable): A string that represents the native LAN Manager type of the client. If SMB_FLAGS2_UNICODE is set in the **Flags2** field of the SMB header of the request, the name string MUST be a null-terminated array of 16-bit Unicode characters. Otherwise, the name string MUST be a null-terminated array of OEM characters. If the name string consists of Unicode characters, this field MUST be aligned to start on a two-byte boundary from the start of the SMB header.<32>

2.2.5 SMB_COM_SESSION_SETUP_ANDX Server Response Extension

An SMB_COM_SESSION_SETUP_ANDX response MUST be sent by a server in reply to a client SMB_COM_SESSION_SETUP_ANDX request (section 2.2.4) when the request is successful.

If extended security is being used, as specified in section 3.2.4.2.3, the response MUST take the following form instead of what is specified in [CIFS] section 4.1.2. Other than the differences in the **WordCount**, **SecurityBlobLength**, and **SecurityBlob** fields, all other fields are as specified in the corresponding response structure in [CIFS] section 4.1.2.



WordCount (1 byte): The word count for this response MUST be 0x04 because there are four 16-bit WORDs between the **WordCount** field and the **ByteCount** field.

AndXCommand (1 byte): The secondary SMB command response in the packet. This value MUST be set to 0xFF if there are no additional SMB command responses in the server response packet. Information about compounded requests is specified in [CIFS] section 3.12.

AndXReserved (1 byte): A reserved field. This field MUST be set to 0 when this response is sent, and the client MUST ignore this value when the message is received. Information about compounded requests is specified in [CIFS] section 3.12.

AndXOffset (2 bytes): This field MUST be set to the offset in bytes from the start of the SMB header to the start of the **WordCount** field in the next SMB command in this packet. The field is valid only if the **AndXCommand** field is not set to 0xFF. Information about compounded requests is specified in [CIFS] section 3.12.

Action (2 bytes): The request mode. If this field is set to 1, the user MUST be logged in as the **guest account**.

SecurityBlobLength (2 bytes): This value MUST specify the length in bytes of the variable length **SecurityBlob** contained within the response.

ByteCount (2 bytes): The number of data bytes in the Data buffer in this packet. If SMB_FLAGS2_UNICODE is set in the **Flags2** field of the SMB header of the request, this field has a minimum value of 6. If SMB_FLAGS2_UNICODE is not set, this field has a minimum value of 3. This field MUST be the total length of the combined **SecurityBlob**, **NativeOS**, **NativeLANMan**, and **PrimaryDomain** fields plus any padding added for alignment.

SecurityBlob (variable): This value MUST contain the authentication token being returned to the client, as specified in section 3.2.4.2.3 and [RFC4178].

NativeOS (variable): A string that represents the native operating system of the server. If SMB_FLAGS2_UNICODE is set in the **Flags2** field of the SMB header of the request, the string MUST be a null-terminated array of 16-bit Unicode characters. Otherwise, the string MUST be a null-terminated array of OEM characters. If the name string consists of Unicode characters, this field MUST be aligned to start on a two-byte boundary from the start of the SMB header.<33>

NativeLANMan (variable): A string that represents the native LAN Manager type of the server. If SMB_FLAGS2_UNICODE is set in the **Flags2** field of the SMB header of the request, the string MUST be a null-terminated array of 16-bit Unicode characters. Otherwise, the string MUST be a null-terminated array of OEM characters. If the name string consists of Unicode characters, this field MUST be aligned to start on a two-byte boundary from the start of the SMB header.<34>

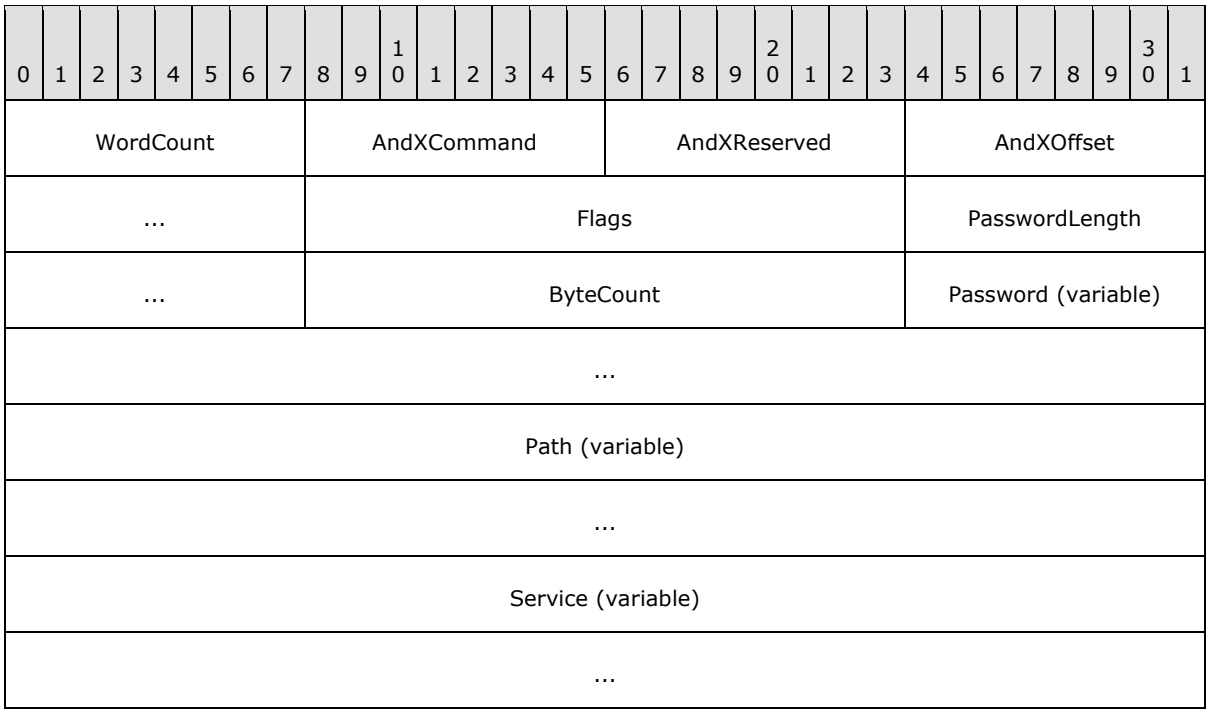
PrimaryDomain (variable): An optional string that MAY represent the primary domain or workgroup name of the server. If SMB_FLAGS2_UNICODE is set in the **Flags2** field of the SMB header of the response, the string MUST be a null-terminated array of 16-bit Unicode characters. Otherwise, the string MUST be a null-terminated array of OEM characters. If the string consists of Unicode characters, this field MUST be aligned to start on a two-byte boundary from the start of the SMB header.

2.2.6 SMB_COM_TREE_CONNECT_ANDX Client Request Extension

An SMB_COM_TREE_CONNECT_ANDX request MUST be sent by a client to establish a tree connection to a share that exists on a server, as specified in [CIFS] section 4.1.4.

The new flags TREE_CONNECT_ANDX_EXTENDED_SIGNATURES and TREE_CONNECT_ANDX_EXTENDED_RESPONSE are used to trigger the new behavior defined in this specification. Other than the new bit values in the **Flags** field of the SMB_COM_TREE_CONNECT_ANDX request, all fields are as specified in the request definition in [CIFS] section 4.1.4.

The **Flags** field MUST be formatted as follows:



WordCount (1 byte): The word count for this request MUST be 0x04 (4) because there are four 16-bit WORDs between the **WordCount** and the **ByteCount** fields.

AndXCommand (1 byte): The secondary SMB command in the packet. This value MUST be set to 0xFF if there are no additional SMB commands in the client request packet. Information about compounded requests is specified in [CIFS] section 3.12.

AndXReserved (1 byte): A reserved field. This MUST be set to 0 when this request is sent, and the server MUST ignore this value when the message is received. Information about compounded requests is specified in [CIFS] section 3.12.

AndXOffset (2 bytes): This field MUST be set to the offset in bytes from the start of the SMB header to the start of the **WordCount** field in the next SMB command in this packet. The field is valid only if the **AndXCommand** field is not set to 0xff. Information about compounded requests is specified in [CIFS] section 3.12.

Flags (2 bytes): A set of options that modify the SMB_COM_TREE_CONNECT_ANDX request. The entire flag set is given here with its symbolic constants. Any combination of the flags below is valid. Any values not given below are considered reserved. The client MUST set them to 0, and the server MUST ignore them.

Value	Meaning
TREE_CONNECT_ANDX_DISCONNECT_TID 0x0001	If set, the tree connect specified by the TID in the SMB header of the request SHOULD be disconnected when the server sends the response. If this tree disconnect fails, the error SHOULD be ignored. This is called bit 0, as specified in [CIFS] section 4.1.4.
TREE_CONNECT_ANDX_EXTENDED_SIGNATURES	If set, the client is requesting signing key

Value	Meaning
0x0004	protection, as specified in section 3.2.4.2.4.
TREE_CONNECT_ANDX_EXTENDED_RESPONSE 0x0008	If set, the client is requesting extended information on the SMB_COM_TREE_CONNECT_ANDX response.

PasswordLength (2 bytes): This field MUST be the length, in bytes, of the **Password** field.

ByteCount (2 bytes): This field MUST be the number of data bytes in the Data buffer in this packet.

Password (variable): An array of ASCII characters that MAY be an authentication response. This is a variable-length field with the length, in bytes, and it MUST be specified by the **PasswordLength** field. For an authentication response, the length specified in the **PasswordLength** field MUST be set to the string size. If authentication is not being used, this field SHOULD be set to a null-terminated array of ASCII characters with the **PasswordLength** field set to the string size, including the terminating null character.

Path (variable): A string that represents the server and share name of the resource to which the client wants to connect. This field SHOULD be encoded using the Universal Naming Convention (UNC) syntax. If SMB_FLAGS2_UNICODE is set in the **Flags2** field of the SMB header of the request, the string MUST be a null-terminated array of 16-bit Unicode characters. Otherwise, the string MUST be a null-terminated array of OEM characters. If the string consists of Unicode characters, this field MUST be aligned to start on a two-byte boundary from the start of the SMB header. A path in UNC syntax would be represented by a string name in the following form:

\\server\share

\\server\share\file<35>

Service (variable): The type of resource the client intends to access. This field MUST be a null-terminated array of ASCII characters even if the client and server have negotiated to use Unicode strings. The valid values for this field are as follows:

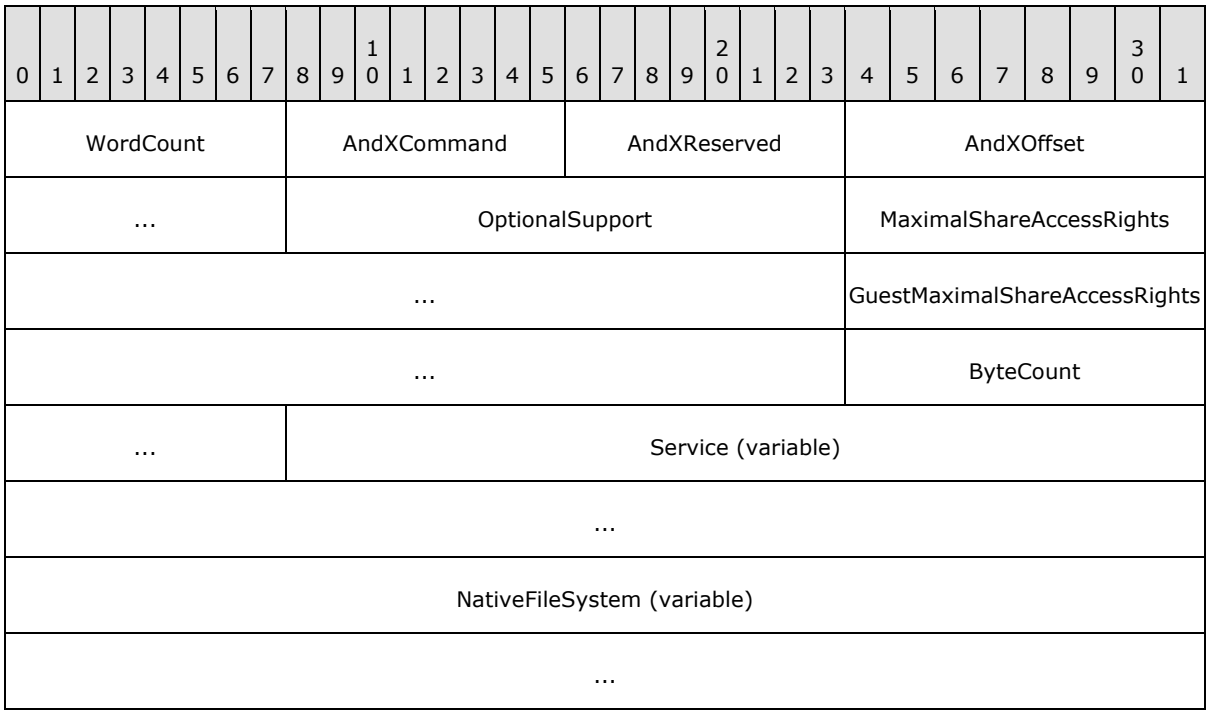
Value	Description	Earliest dialect allowed
A:	Disk share	PC NETWORK PROGRAM 1.0
LPT1:	Printer	PC NETWORK PROGRAM 1.0
IPC	Named pipe	MICROSOFT NETWORKS 3.0
COMM	Communication device	MICROSOFT NETWORKS 3.0
?????	Matches any type of device or resource	MICROSOFT NETWORKS 3.0

2.2.7 SMB_COM_TREE_CONNECT_ANDX Server Response Extension

An SMB_COM_TREE_CONNECT_ANDX response MUST be sent by a server in reply to a client SMB_COM_TREE_CONNECT_ANDX request when the request was successful.

When a server returns extended information, the response takes the format defined below. Aside from the **WordCount**, **MaximalShareAccessRights**, **GuestMaximalAccessRights** fields, the new

OptionalSupport flags and **NativeFileSystem** fields are as specified in the corresponding response structure in [CIFS] section 4.1.4.



WordCount (1 byte): The word count for this response MUST be 0x07 because there are seven 16-bit WORDs between the **WordCount** field and the **ByteCount** field.

AndXCommand (1 byte): The secondary SMB command response in the packet. This value MUST be set to 0xFF if there are no additional SMB command responses in the server response packet. Information about compounded requests is specified in [CIFS] section 3.12.

AndXReserved (1 byte): A reserved field. This MUST be set to 0 when this response is sent and the client MUST ignore this value when the message is received. Information about compounded requests is specified in [CIFS] section 3.12.

AndXOffset (2 bytes): MUST be set to the offset in bytes from the start of the SMB header to the start of the **WordCount** field in the next SMB command in this packet. The field is only valid if the **AndXCommand** field is not set to 0xFF. Information about compounded requests is specified in [CIFS] section 3.12.

OptionalSupport (2 bytes): The following new **OptionalSupport** field flags are the new extensions added to the CIFS protocol: SMB_CSC_MASK, SMB_UNIQUE_FILE_NAME, and SMB_EXTENDED_SIGNATURES. The table is given below for completeness, indicating the supported bits and their associated symbolic constants. SMB_CSC_MASK MUST be a combination of two bits used to identify one of four possibilities, as specified in the second table in this section. Any combination of the flags below MUST be supported. All undefined values are considered reserved. the server SHOULD set them to 0, and the client MUST ignore them.

Value	Meaning
SMB_SUPPORT_SEARCH_BITS 0x0001	If set, the server supports the use of SearchAttributes in client requests, as specified in [CIFS] sections 4.2.10, 4.2.11, 5.7, 5.8, and 5.15.
SMB_SHARE_IS_IN_DFS 0x0002	If set, this share is managed by DFS, as specified in [MS-DFSC].
SMB_CSC_MASK 0x000C	Offline-caching bits for this share. These bits are interpreted as specified in the next table in this section.
SMB_UNIQUE_FILE_NAME 0x0010	If set, the server is using long file names only and does not support short file names. If set, the server permits the client to cache the results of a directory enumeration on this share. The client MAY choose to satisfy file attribute queries from its cache, and thus could present a slightly stale view of files on the share. The client MUST NOT cache remote file system information for more than 60 seconds.<36><37>
SMB_EXTENDED_SIGNATURES 0x0020	If set, the server is using signing key protection as the client requested.

The two bits referenced by the SMB_CSC_MASK MUST be interpreted as follows:

Value	Meaning
SMB_CSC_CACHE_MANUAL_REINT 00	Clients are allowed to cache files that the user requests for offline use, but there is no automatic file-by-file reintegration.
SMB_CSC_CACHE_AUTO_REINT 01	Clients are allowed to automatically cache the files that a user or application modifies for offline use, and automatic file-by-file reintegration MUST be permitted.
SMB_CSC_CACHE_VDO 10	Clients are allowed to automatically cache the files that a user or application modifies for offline use, and clients are permitted to work from their local cache even while online.
SMB_CSC_NO_CACHING 11	No offline caching is allowed for this share.

MaximalShareAccessRights (4 bytes): This field MUST specify the maximum rights the user has to this share based on the security enforced by the share. This value is as specified in the ACCESS_MASK format, as specified in [CIFS] section 3.7.

GuestMaximalShareAccessRights (4 bytes): This field MUST specify the maximum rights the guest account has on this share based on the security enforced by the share. This value is as specified in the ACCESS_MASK format, as specified in [CIFS] section 3.7. Note that the notion of a guest account is implementation specific. Implementations that do not support the notion of a guest account MUST set this field to zero implying no access.<38>

ByteCount (2 bytes): This field MUST specify the number of data bytes in the Data buffer in this packet.

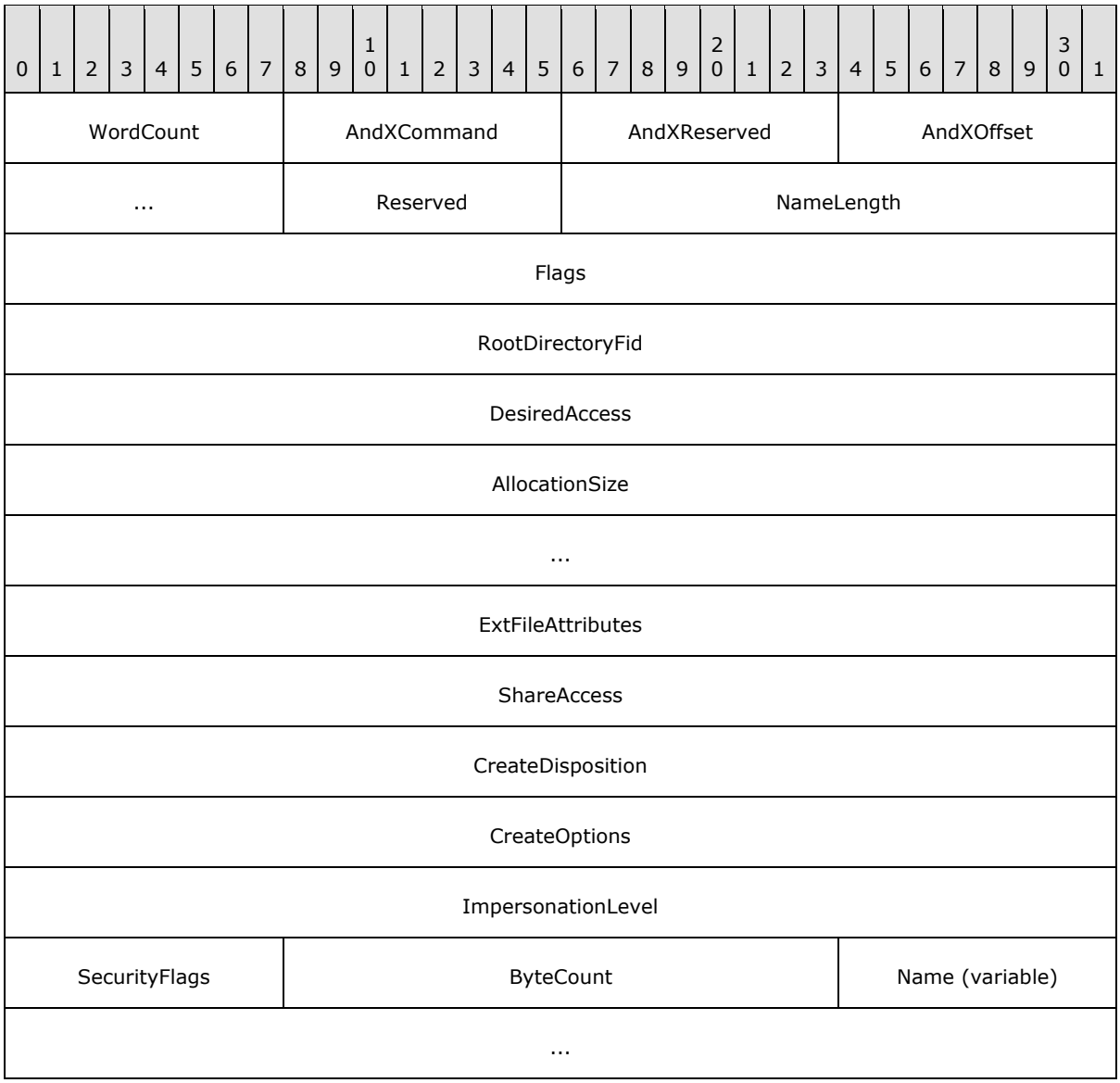
Service (variable): The **Service** field indicates the type of resource the client is accessing. The **Service** field MUST be encoded as a null-terminated array of ASCII characters, even if the client and server have negotiated to use Unicode strings. The valid values for this field are as follows:

Value	Meaning
"A:"	Disk share
"LPT1:"	Printer
"IPC"	Named pipe
"COMM"	Communication device

NativeFileSystem (variable): The name of the file system on the local resource that is being connected to. If the **Flags2** field in the SMB_HEADER includes SMB_FLAGS2_UNICODE, this value MUST be a null-terminated string of Unicode characters. Otherwise, this field MUST be a null-terminated string of ASCII characters. For resources that are not backed by a file system, such as the IPC$ share used for named pipes, this field MUST be set to a single null character.

2.2.8 SMB_COM_NT_CREATE_ANDX Client Request Extension

An SMB_COM_NT_CREATE_ANDX request is sent by a client to open a file or named pipe on the target server, as specified in [CIFS] section 4.2.1. The new value NT_CREATE_REQUEST_EXTENDED_RESPONSE in the **Flags** field of the SMB_COM_NT_CREATE_ANDX request is used to trigger the new behavior defined in this specification. All other fields match their descriptions in the request structure as specified in [CIFS] section 4.2.1.



WordCount (1 byte): The word count for this request MUST be 0x18 (24), because there are 24 16-bit WORDs between the **WordCount** and the **ByteCount** fields.

AndXCommand (1 byte): The secondary SMB command in the packet. This value MUST be set to 0xFF if there are no additional SMB commands in the client request packet. Information about compounded requests is specified in [CIFS] section 3.12.

AndXReserved (1 byte): A reserved field. This MUST be set to 0 when this request is sent, and the server MUST ignore this value when the message is received. Information about compounded requests is specified in [CIFS] section 3.12.

AndXOffset (2 bytes): This field MUST be set to the offset in bytes from the start of the SMB header to the start of the **WordCount** field in the next SMB command in this packet. The field is only valid if the **AndXCommand** field is not set to 0xFF. Information about compounded requests is specified in [CIFS] section 3.12.

Reserved (1 byte): An unused value that SHOULD be set to 0 when sending this message. The server MUST ignore this field when receiving this message.

NameLength (2 bytes): This field MUST be the length, in bytes, of the **Name** field.

Flags (4 bytes): A set of flags that modify the client request. Unused bit fields SHOULD be set to 0 by the server when sending a response, and MUST be ignored when received by the client. The **Flags** field in the SMB_COM_NT_CREATE_ANDX request MUST be used as follows.

Value	Meaning
NT_CREATE_REQUEST_OPLOCK 0x00000002	If set, the client is requesting an oplock. This flag is as specified in [CIFS] section 4.2.1.
NT_CREATE_REQUEST_OPBATCH 0x00000004	If set, the client is requesting a batch oplock. This flag is as specified in [CIFS] section 4.2.1.
NT_CREATE_OPEN_TARGET_DIR 0x00000008	If set, the client indicates that the target of the open is a directory. This flag is as specified in [CIFS] section 4.2.1.
NT_CREATE_REQUEST_EXTENDED_RESPONSE 0x00000010	If set, the client is requesting extended information in the response.

RootDirectoryFid (4 bytes): If non-zero, this value is the Fid of an opened root directory, and the **Name** field MUST be handled as relative to the directory specified by this Fid.

DesiredAccess (4 bytes): Access wanted. This value MUST be specified in the ACCESS_MASK format, as specified in [CIFS] section 3.7. If no value is specified for this field, it still allows a client to query attributes without actually accessing the file.

AllocationSize (8 bytes): The client MUST set this value to the initial allocation size of the file in bytes. The server MUST ignore this field if this request is to open an existing file. In other words, this field MUST be used only if the file is created, overwritten, or superseded. The client MUST be set to 0 in all other cases.

ExtFileAttributes (4 bytes): Extended attributes and flags for this file or directory. The field MUST be specified in extended file attribute encoding format, as specified in [CIFS] section 3.11.

ShareAccess (4 bytes): Type of shared access requested for this file or directory. If this field is set to 0, other clients MUST NOT be allowed to open the file or directory for read, write, or delete. This field MUST be set to 0 or a bitwise OR of the following possible values. The values below are in little-endian format.<39>

Value	Meaning
FILE_SHARE_READ 0x00000001	Other open operations can be performed on this file for read access.
FILE_SHARE_WRITE 0x00000002	Other open operations can be performed on this file for write access.
FILE_SHARE_DELETE 0x00000004	Other open operations can be performed on this file for delete access.

CreateDisposition (4 bytes): The action to take if a file does or does not exist. This field MUST be set to one of the following values.<40>

Value	Meaning
FILE_SUPERSEDE 0x00000000	If the file already exists, replace it with the file. If the file does not exist, create the file.
FILE_OPEN 0x00000001	If the file already exists, open the file instead of creating a new file. If the file does not already exist, fail the request, and do not create a new file.
FILE_CREATE 0x00000002	If the file already exists, fail the request, and do not create or open the file. If the file does not already exist, create the file.
FILE_OPEN_IF 0x00000003	If the file already exists, open the file. If the file does not already exist, create the file.
FILE_OVERWRITE 0x00000004	If the file already exists, open the file and overwrite it. If the file does not already exist, fail the request.
FILE_OVERWRITE_IF 0x00000005	If the file already exists, open the file and overwrite it. If the file does not already exist, create the file.

CreateOptions (4 bytes): The options to use if creating the file or directory. This field MUST be set to zero or a combination of the following possible values. Unused bit fields SHOULD be set to 0 by the server when sending a response, and MUST be ignored when received by the client.<41>

Value	Meaning
FILE_DIRECTORY_FILE 0x00000001	The file being created or opened is a directory file. With this option, the **CreateDisposition** field MUST be set to FILE_CREATE, FILE_OPEN, or FILE_OPEN_IF. When this bit field is set, other compatible CreateOptions include only the following: FILE_WRITE_THROUGH, FILE_OPEN_FOR_BACKUP_INTENT, and FILE_OPEN_BY_FILE_ID.
FILE_WRITE_THROUGH 0x00000002	Applications that write data to the file MUST actually transfer the data into the file before any write request is considered complete. This option SHOULD be automatically set by the server if FILE_NO_INTERMEDIATE_BUFFERING is set.
FILE_SEQUENTIAL_ONLY 0x00000004	This option indicates that access to the file may be sequential. The server MAY use this information to influence its caching and read-ahead strategy for this file. The file MAY in fact be accessed randomly, but the server MAY optimize its caching and read-ahead policy for sequential access.
FILE_NO_INTERMEDIATE_BUFFERING 0x00000008	The file SHOULD NOT be cached or buffered in an internal buffer by the server. This option is incompatible when the FILE_APPEND_DATA bit field is set in the **DesiredAccess** field.
FILE_SYNCHRONOUS_IO_ALERT 0x00000010	This flag MUST be ignored by the server, and clients SHOULD set this to 0.<42>

Value	Meaning
FILE_SYNCHRONOUS_IO_NONALERT 0x00000020	This flag MUST be ignored by the server, and clients SHOULD set this to 0.<43>
FILE_NON_DIRECTORY_FILE 0x00000040	The server MUST fail the request if the file being opened is a directory.
FILE_CREATE_TREE_CONNECTION 0x00000080	This option SHOULD NOT be sent by the clients, and this option MUST be ignored by the server.
FILE_COMPLETE_IF_OPLOCKED 0x00000100	This option SHOULD NOT be sent by the clients, and this option MUST be ignored by the server.
FILE_NO_EA_KNOWLEDGE 0x00000200	The application that initiated the clients request does not understand extended attributes. If the extended attributes on an existing file being opened indicate that the caller SHOULD understand EAs to properly interpret the file, the server SHOULD fail this request.
FILE_OPEN_FOR_RECOVERY 0x00000400	This option SHOULD NOT be sent by the clients, and this option MUST be ignored if received by the server.
FILE_RANDOM_ACCESS 0x00000800	Indicates that access to the file may be random. The server MAY use this information to influence its caching and read-ahead strategy for this file. This is a hint to the server that sequential read-ahead operations may not be appropriate on the file.
FILE_DELETE_ON_CLOSE 0x00001000	The file SHOULD be automatically deleted when the last open request on this file is closed. When this option is set, the **DesiredAccess** field MUST include the DELETE flag. This option is often used for temporary files.
FILE_OPEN_BY_FILE_ID 0x00002000	Open a file based on the FileId. The server MUST fail the request if this option is set.
FILE_OPEN_FOR_BACKUP_INTENT 0x00004000	The file is being opened or created for the purposes of either a backup or a restore operation. Thus, the server MAY make appropriate checks to ensure that the caller is capable of overriding whatever security checks have been placed on the file to allow a backup or restore operation to occur. The server MAY choose to check for certain access rights to the file before checking the **DesiredAccess** field.
FILE_NO_COMPRESSION 0x00008000	When a new file is created, the file MUST not be compressed even it is on a compressed volume. The flag MUST be ignored when opening an existing file.
FILE_RESERVE_OPFILTER 0x00100000	This option SHOULD NOT be sent by the clients, and this option MUST be ignored if received by the server.
FILE_OPEN_REPARSE_POINT 0x00200000	If the file or directory being opened is a **reparse point**, this option requests that the server open the reparse point itself, rather than the target to which the reparse point points.
FILE_OPEN_NO_RECALL 0x00400000	In a hierarchical storage management (HSM) environment, this option requests that the file SHOULD NOT be recalled from tertiary storage such as tape. A file

Value	Meaning
	recall can take up to several minutes in an HSM environment. Therefore, the clients can specify this option to avoid delays.
FILE_OPEN_FOR_FREE_SPACE_QUERY 0x00800000	This option SHOULD NOT be sent by the clients, and this option MUST be ignored if received by the server.

ImpersonationLevel (4 bytes): This field specifies the information given to the server about the client and how the server MUST represent, or impersonate, the client. Security impersonation levels govern the degree to which a server process can act on behalf of a client process. This field MUST be set to one of the following values.<44>

Value	Meaning
SECURITY_ANONYMOUS 0x00000000	The server cannot obtain identification information about the client, and it cannot impersonate the client.
SECURITY_IDENTIFICATION 0x00000001	The server can obtain security information about the client, such as **security identifiers (SIDs)** and privileges, but it cannot impersonate the client. This is useful for servers that export their own objects; for example, database products that export tables and views. Using the retrieved client-security information, the server can make access-validation decisions without being capable of using other services that are using the client's security context.
SECURITY_IMPERSONATION 0x00000002	The server can impersonate the client's security context on the server system. The server cannot impersonate the client on other remote systems.
SECURITY_DELEGATION 0x00000003	The server can impersonate the client's security context on the server system and on remote systems.

If the upper-layer protocols do not specify any of the previous values, the SMB client MUST default to SECURITY_IMPERSONATION.

SecurityFlags (1 byte): A set of options that specify the security tracking mode. These options specify whether the server is to be given a **snapshot** of the client's security context (called static tracking), or is to be continually updated to track changes to the client's security context (called dynamic tracking). When bit 0 of the **SecurityFlags** field is set to FALSE, static tracking is requested. When bit 0 the **SecurityFlags** field is set to TRUE, dynamic tracking is requested. Unused bit fields SHOULD be set to 0 by the client when sending a request, and MUST be ignored when received by the server. This field MUST be set to 0 or a combination of the following possible values.<45>

Value	Meaning
SMB_SECURITY_CONTEXT_TRACKING 0x00000001	When this bit field is set, dynamic tracking is requested. When this bit field is not set, static tracking is requested.
SMB_SECURITY_EFFECTIVE_ONLY 0x00000002	When this bit field is set, the server MAY enable or disable privileges and groups that the client's security context MAY include.

ByteCount (2 bytes): The number of data bytes in the Data buffer in this packet. If SMB_FLAGS2_UNICODE is set in the **Flags2** field of the SMB header of the request, this field

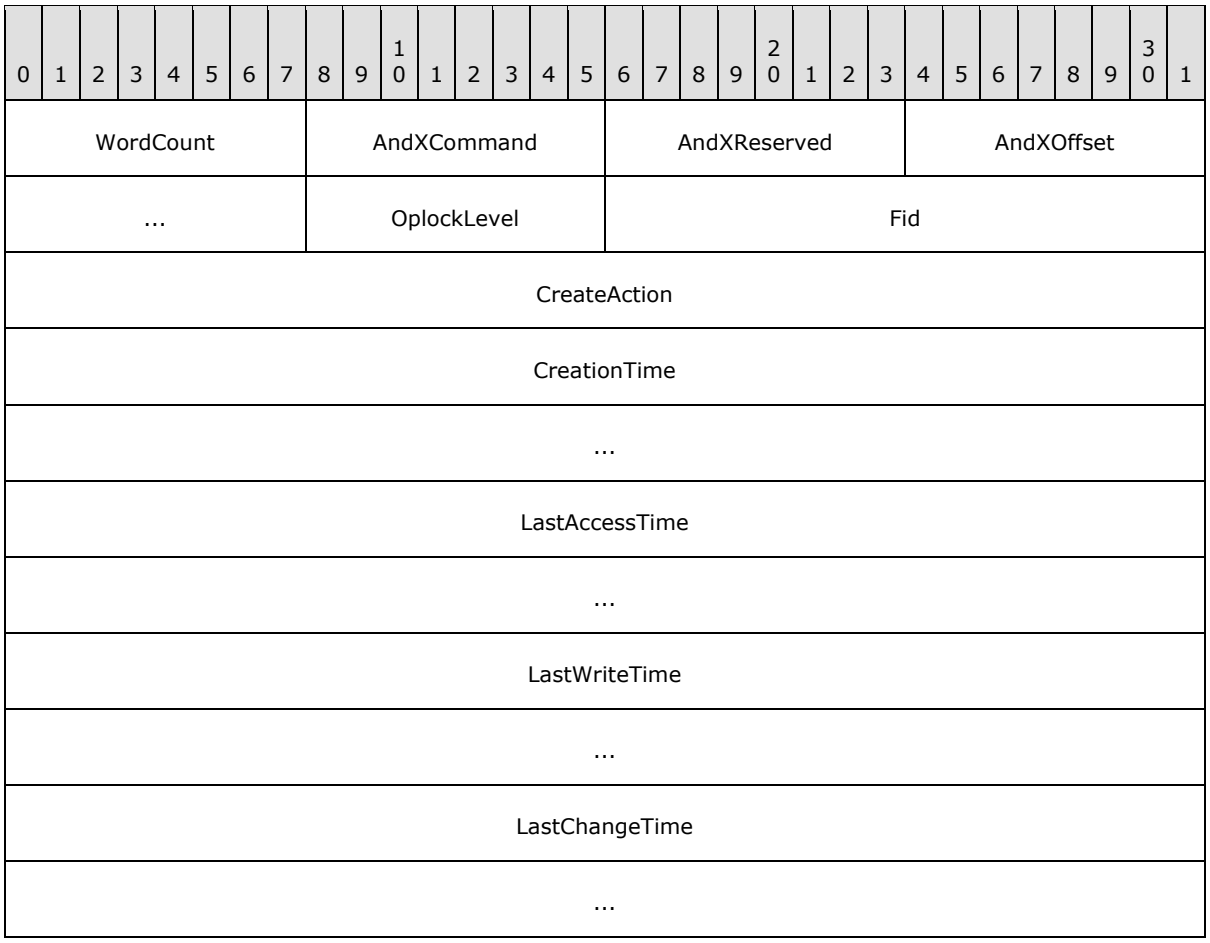
has a minimum value of 3. If SMB_FLAGS2_UNICODE is not set, this field has a minimum value of 2. This field MUST be the total length of the **Name** field plus any padding added for alignment.

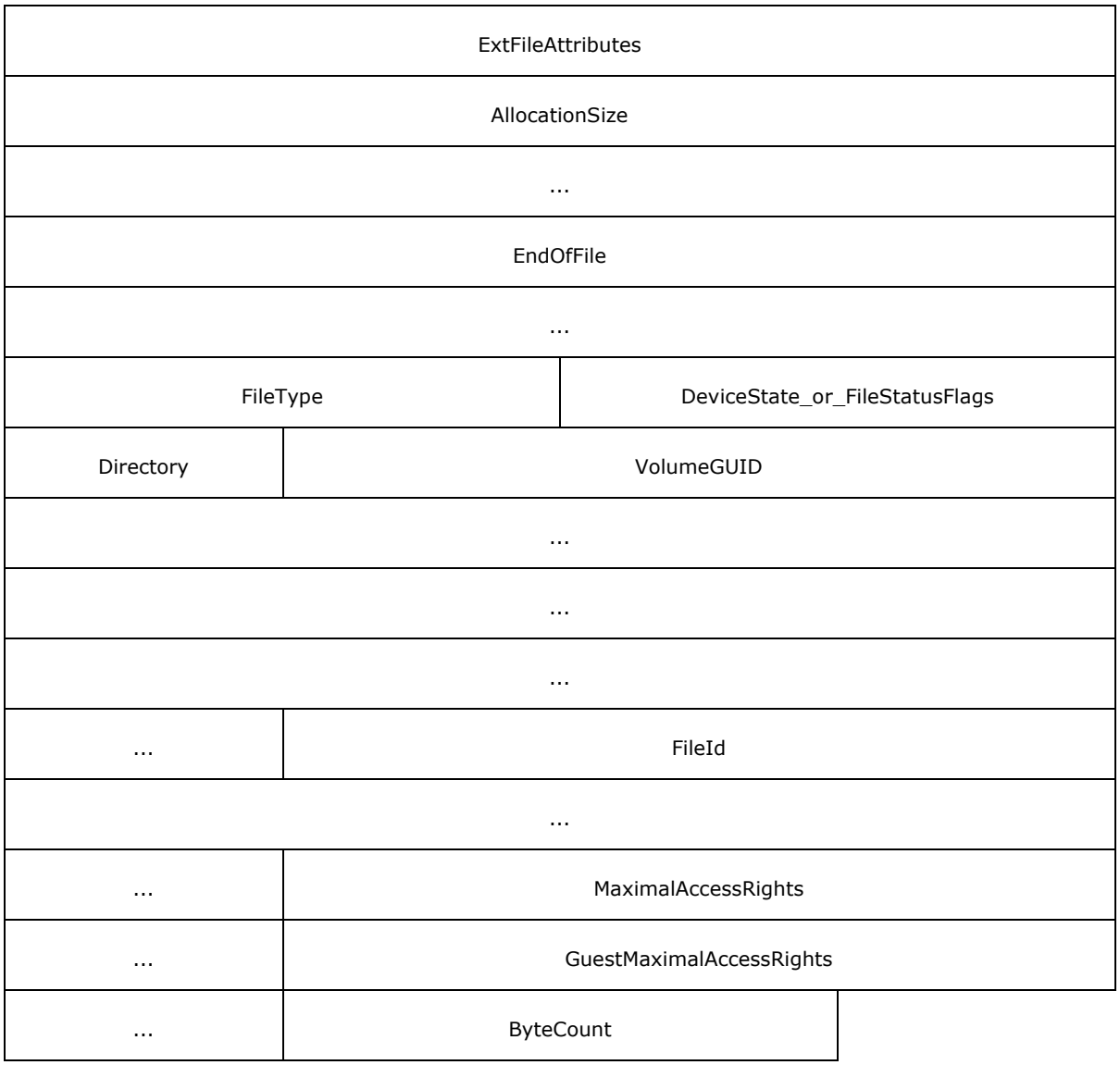
Name (variable): A string that represents the name of the file to open or create on the server. If SMB_FLAGS2_UNICODE is set in the **Flags2** field of the SMB header of the response, the Name string MUST be a null-terminated array of 16-bit Unicode characters. Otherwise, the Name string MUST be a null-terminated array of ASCII characters. If the Name string consists of Unicode characters, this field MUST be aligned to start on a two-byte boundary from the start of the SMB header.

2.2.9 SMB_COM_NT_CREATE_ANDX Server Response Extension

An SMB_COM_NT_CREATE_ANDX response MUST be sent by a server in reply to a client SMB_COM_NT_CREATE_ANDX request when the request is successful, as specified in [CIFS] section 4.2.1.<46>

When a client requests extended information, the response takes the form described below. Aside from the **WordCount**, **FileStatusFlags**, **FileId**, **VolumeGUID**, **MaximalAccessRights**, and **GuestMaximalAccessRights** fields, all other fields are described in the corresponding response structure, as specified in [CIFS] section 4.2.1.





WordCount (1 byte): The word count for this response MUST be 0x2A (42). **WordCount** in this case is not used as the count of parameter words; instead it is just a number.

AndXCommand (1 byte): The secondary SMB command response in the packet. This value MUST be set to 0xFF if there are no additional SMB command responses in the server response packet. Information about compounded requests is specified in [CIFS] section 3.12.

AndXReserved (1 byte): A reserved field. This MUST be set to 0 when this response is sent, and the client MUST ignore this value when the message is received. Information about compounded requests is specified in [CIFS] section 3.12.

AndXOffset (2 bytes): This field MUST be set to the offset in bytes from the start of the SMB header to the start of the **WordCount** field in the next SMB command in this packet. The field is only valid if the **AndXCommand** field is not set to 0xFF. Information about compounded requests is specified in [CIFS] section 3.12.

OplockLevel (1 byte): The oplock level granted.

This field MUST be interpreted as follows.<47>

Value	Meaning
SMB_OPLOCK_LEVEL_NONE 0x00	Oplock not granted.
SMB_OPLOCK_LEVEL_EXCLUSIVE 0x01	Exclusive oplock granted.
SMB_OPLOCK_LEVEL_BATCH 0x02	Batch oplock granted.
SMB_OPLOCK_LEVEL_II 0x03	Level II oplock granted.

Fid (2 bytes): The SMB file identifier returned by the SMB server for the file or device that was opened or created.

CreateAction (4 bytes): The action taken. This field MUST be interpreted as follows.

Value	Meaning
FILE_SUPERSEDED 0x0000	The file or directory has been superseded.
FILE_OPENED 0x0001	The file or directory has been opened.
FILE_CREATED 0x0002	The file or directory has been created.
FILE_OVERWRITTEN 0x0003	The file or directory has been overwritten.
FILE_EXISTS 0x0004	The file already exists.
FILE_DOES_NOT_EXIST 0x0005	The file does not exist.

CreationTime (8 bytes): The time when the file or named pipe was created or opened. This field MUST be specified in the TIME (**MS-DTYP** section **2.2.44**) encoding format, as specified in [CIFS] section 3.5.

LastAccessTime (8 bytes): Time when the file or named pipe was last accessed. This field MUST be specified in the TIME (**MS-DTYP** section **2.2.44**) encoding format, as specified in [CIFS] section 3.5.

LastWriteTime (8 bytes): Time when the file or named pipe was last written to. This field MUST be specified in the TIME (**MS-DTYP** section **2.2.44**) encoding format, as specified in [CIFS] section 3.5.

LastChangeTime (8 bytes): Time when the file or named pipe was last changed. This field MUST be specified in the TIME (**MS-DTYP** section **2.2.44**) encoding format, as specified in [CIFS] section 3.5.

ExtFileAttributes (4 bytes): Extended attributes and flags for this file or directory. The field MUST be specified in extended file attribute encoding format, as specified in [CIFS] section 3.11.

AllocationSize (8 bytes): This field MUST specify the number of bytes allocated for the file. This field is encoded as a LARGE_INTEGER, as specified in **MS-DTYP** section **2.3.3**.<48>

EndOfFile (8 bytes): MUST specify the end of file offset in bytes. This field is encoded as a LARGE_INTEGER, as specified in **MS-DTYP** section **2.3.3**.

FileType (2 bytes): The file type. This field MUST be interpreted as follows.

Value	Meaning
FileTypeDisk 0x0000	Disk file
FileTypeByteModePipe 0x0001	Byte-mode named pipe
FileTypeMessageModePipe 0x0002	Messagemode named pipe
FileTypePrinter 0x0003	Printer
FileTypeCommDevice 0x0004	Communications device
FileTypeUnknown 0xFFFF	Unknown file type

DeviceState_or_FileStatusFlags (2 bytes): A union. If the **FileType** field is a named pipe, this field MUST be the DeviceState of the named pipe. If the **FileType** field is a file or directory, this field MUST be the **FileStatusFlags** field that returns extra information about the file or directory.

Any combination of the following bits MUST be valid for the **FileStatusFlags** field. Unused bit fields SHOULD be set to 0 by the server when sending a response, and MUST be ignored when received by the client.

Value	Meaning
NO_REPARSETAG 0x0004	The file or directory is not a reparse point.
NO_SUBSTREAMS 0x0002	The file or directory has no other streams of data aside from the main data stream.
NO_EAS 0x0001	The file or directory has no extended attributes.

Any combination of the following bits MUST be valid for the **DeviceState** field. Unused bit fields SHOULD be set to 0 by the server when sending a response, and MUST be ignored when received by the client.

Value	Meaning
ICount 0x00FF	8-bit count to control pipe instancing.
ReadMode 0x0100	This bit field indicates the client read mode for the named pipe. If set, it indicates that the client opened the named pipe or set the named pipe to **message mode**. If not set, the named pipe was opened or set to byte stream mode. This bit field has no effect on writes to the named pipe.
NamedPipeType 0x0400	This bit field indicates the type of the named pipe when the named pipe was created by the server. If set, the named pipe was created by the server as a message mode pipe. If not set, the named pipe was created as a stream mode pipe.
EndPoint 0x4000	A value that indicates the endpoint of the pipe. A value of 0 indicates the client end of the pipe. A value of 1 indicates the server end of the pipe. The SMB server MUST set the EndPoint bit to 0 when responding to the client request because the endpoint specified is the client end of the pipe.
Blocking 0x8000	If set, indicates that named pipe read requests return immediately if no data is available, and writes return immediately. If not set, named pipe read requests block if no data is available, and writes block until their data is consumed by a read.

Directory (1 byte): A value that indicates if this is a directory. MUST be non-zero when this is a directory.

VolumeGUID (16 bytes): A unique value that identifies the volume on which the file resides. This field MUST contain 0 if the underlying file system does not support volume GUIDs.

FileId (8 bytes): This field MUST be a 64-bit opaque value that uniquely identifies this file on a volume. This field MUST contain 0 if the underlying file system does not support unique FileId numbers on a volume.<49>

MaximalAccessRights (4 bytes): Maximum access rights that the user opening the file has for this file. This value MUST be specified according to the ACCESS_MASK format, as specified in [CIFS] section 3.7.

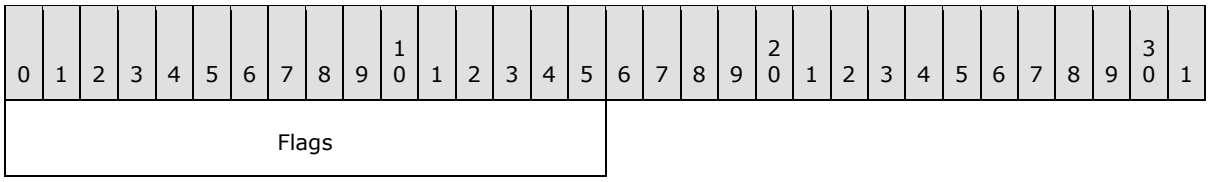
GuestMaximalAccessRights (4 bytes): The maximum access rights that the guest account has when opening this file. This value MUST be specified according to the ACCESS_MASK format, as specified in [CIFS] section 3.7. Note that the notion of a guest account is somewhat implementation specific. Implementations that do not support the notion of a guest account MUST set this field to 0.<50>

ByteCount (2 bytes): This value MUST be the number of bytes that follows the **ByteCount** field. The server MUST set this value to 0 if the underlying file system does not support volume GUIDs.

2.2.10 SMB_COM_OPEN_ANDX Client Request Extension

An SMB_COM_OPEN_ANDX request is sent by a client to open a file or named pipe on a server, as specified in [CIFS] section 5.8. The new value, SMB_OPEN_EXTENDED_RESPONSE, in the **Flags** field of the SMB_COM_OPEN_ANDX request is used to trigger the new behavior specified in this specification. All other fields match their description in the request structure, as specified in [CIFS] section 5.8.

The **Flags** field in the SMB_COM_OPEN_ANDX request is used as follows.



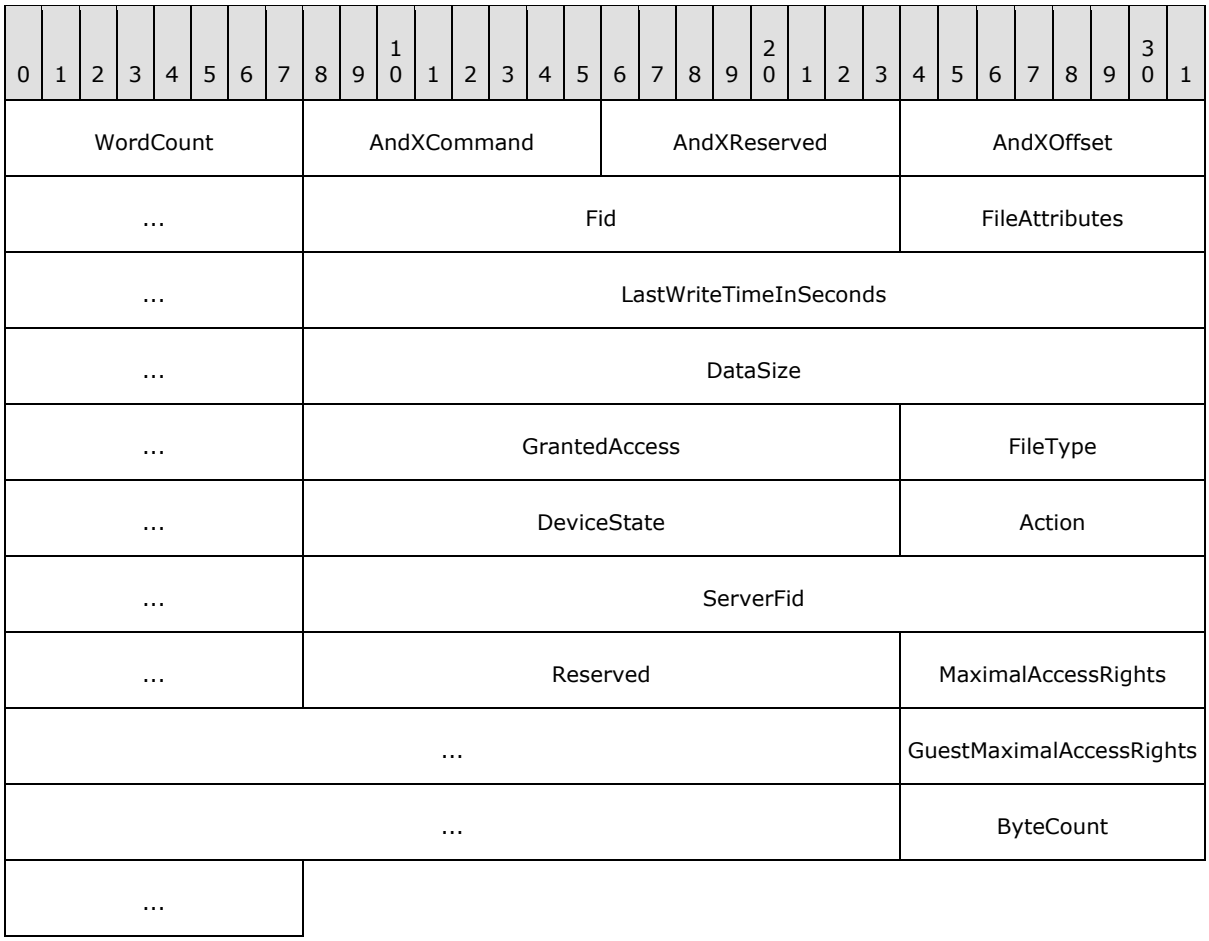
Flags (2 bytes): A 16 bit field of bit flags. For completeness, all flags are listed in the following table with their symbolic constants. Any values not listed are considered reserved, and SHOULD be set to 0 by the client, and MUST be ignored by the server.

Value	Meaning
SMB_OPEN_EXTENDED_RESPONSE 0x0010	If set, the client is requesting extended information in the response.
SMB_OPEN_OPBATCH 0x0004	If set, the client is requesting a batch oplock. This flag is specified in [CIFS] section 5.8 as bit 2.
SMB_OPEN_OPLOCK 0x0002	If set, the client is requesting an oplock. This flag is specified in [CIFS] section 5.8 as bit 1.
SMB_OPEN_QUERY_INFORMATION 0x0001	If set, the client is requesting additional info in the response. The server MUST set **DataSize**, **FileAttributes**, **GrantedAccess**, **FileType**, and **DeviceState** in the response. If not set, the server MUST set these fields to 0. This flag is specified in [CIFS] section 5.8 as bit 0.

2.2.11 SMB_COM_OPEN_ANDX Server Response Extension

An SMB_COM_OPEN_ANDX response MUST be sent by a server in reply to a client SMB_COM_OPEN_ANDX request when the request is successful.

If the client requested extended information, the response takes the format defined below. Aside from **WordCount**, **MaximalAccessRights**, and **GuestMaximalAccessRights** fields, all other fields are identical to the corresponding response packet, as specified in [CIFS] section 5.8.



WordCount (1 byte): The word count for this response MUST be 0x13 (19) because there are nineteen 16-bit WORDs between the **WordCount** field and the **ByteCount** field.

AndXCommand (1 byte): The secondary SMB command response in the packet. This value MUST be set to 0xFF if there are no additional SMB command responses in the server response packet. Information about compounded requests is specified in [CIFS] section 3.12.

AndXReserved (1 byte): A reserved field. This MUST be set to 0 when this response is sent, and the client MUST ignore this value when the message is received. Information about compounded requests is specified in [CIFS] section 3.12.

AndXOffset (2 bytes): This field MUST be set to the offset in bytes from the start of the SMB header to the start of the **WordCount** field in the next SMB command in this packet. The field is only valid if the **AndXCommand** field is not set to 0xFF. Information about compounded requests is specified in [CIFS] section 3.12.

Fid (2 bytes): This field MUST be the SMB file identifier returned by the SMB server for the file or device that was opened or created.

FileAttributes (2 bytes): File attributes. This value MUST be specified according to the file attribute encoding format, as specified in [CIFS] section 3.10.

LastWriteTimeInSeconds (4 bytes): The time when the file or named pipe was last written. This value MUST be rounded up to the last two seconds. This field MUST be specified in the UTIME encoding format, as specified in [CIFS] section 3.5.

DataSize (4 bytes): The current file size in bytes.

GrantedAccess (2 bytes): The access permissions currently allowed. This value MUST be specified according to the ACCESS MODE ENCODING format, as specified in [CIFS] section 3.6.

FileType (2 bytes): The file type. This field MUST be interpreted as follows.

Value	Meaning
FileTypeDisk 0x0000	Disk file
FileTypeByteModePipe 0x0001	Byte mode named pipe
FileTypeMessageModePipe 0x0002	Message mode named pipe
FileTypePrinter 0x0003	Printer
FileTypeCommDevice 0x0004	Communications device
FileTypeUnknown 0xFFFF	Unknown file type

DeviceState (2 bytes): The **pipe state** for the named pipe; that is, a series of attributes that describes how the named pipe interacts for various I/O operations and indicates how much data is currently available for reading from the named pipe. If the **FileType** field was not set to a named pipe (FileTypeByteModePipe or FileTypeMessageModePipe), this field SHOULD be set to 0 by the server, and MUST be ignored when received by the client. For a named pipe, this field MUST be as follows. Unused bit fields SHOULD be set to 0 by the server when sending a response, and MUST be ignored when received by the client.

Value	Meaning
ICount 0x00FF	An 8-bit count to control pipe instancing.
ReadMode 0x0100	This bit field indicates the client read mode for the named pipe. If set, it indicates that the client opened the named pipe or set the named pipe to message mode. If not set, the named pipe was opened or set to byte stream mode. This bit field has no effect on writes to the named pipe.
NamedPipeType 0x0400	This bit field indicates the type of the named pipe when the named pipe was created by the server. If set, the named pipe was created by the server as a message mode pipe. If not set, the named pipe was created as a stream mode pipe.
EndPoint 0x4000	A value that indicates the endpoint of the pipe. A value of 0 indicates the client end of the pipe. A value of one indicates the server end of the pipe. The SMB server MUST set the **EndPoint** bit to 0 when responding to the client request

Value	Meaning
	because the endpoint specified is the client end of the pipe.
Blocking 0x8000	If set, this value indicates that named pipe read requests return immediately if no data is available, and writes return immediately. If not set, named pipe read requests block if no data is available, and writes block until their data is consumed by a read.

Action (2 bytes): The action taken. This field MUST be interpreted as follows.

Value	Meaning
SMB_OACT_OPENED 0x01	The file existed and was opened.
SMB_OACT_CREATED 0x02	The file did not exist but was created.
SMB_OACT_TRUNCATED 0x03	The file existed and was truncated.

The value 0x8000 (SMB_OACT_OPLOCK) can be combined with the above values. When this value is set in the **Action** field, this indicates that an opportunistic lock was granted.

ServerFid (4 bytes): An optional 32-bit server file identifier that MUST uniquely identify the opening of the file on the server.<51>

An unused value that SHOULD be set to 0 when sending this message. The server MUST ignore this field when receiving this message.

MaximalAccessRights (4 bytes): The maximum access rights that this user has on this file. This field MUST be encoded in the ACCESS_MASK format, as specified in [CIFS] section 3.7.

GuestMaximalAccessRights (4 bytes): The maximum access rights that the guest account has on this file. This field MUST be encoded in the ACCESS_MASK format, as specified in [CIFS] section 3.7. Note that the notion of a guest account is somewhat implementation specific. Implementations that do not support the notion of a guest account MUST set this field to 0.<52>

ByteCount (2 bytes): This value MUST be the number of bytes that follows the **ByteCount** field. The server MUST set this value to 0.

2.2.12 SMB_COM_TRANSACTION Extensions

The SMB_COM_TRANSACTION request MUST be sent by a client to execute a named pipe or mailslot operation on a server. The base SMB_COM_TRANSACTION format and base SMB_COM_TRANSACTION message sequencing and handling are as specified in [CIFS] section 3.13, but the actual subcommand codes for handling named pipes and mailslots are not detailed as part of that reference.

The command being executed by a transaction is determined by the **SetupCount**, the subcommand (first Setup WORD), and the **Name** field information in the SMB_COM_TRANSACTION request, as specified in [CIFS] section 3.13.1.

Named pipe operations MUST have a **SetupCount** of 2, except for TRANS_WAIT_NMPIPE, which MUST have a **SetupCount** of 1. If the first Setup WORD subcommand code is not listed below, it MUST fail the request with STATUS_INVALID_PARAMETER.

Named pipe subcommand codes	Value
TRANS_SET_NMPIPE_STATE	0x0001
TRANS_RAW_READ_NMPIPE	0x0011
TRANS_QUERY_NMPIPE_STATE	0x0021
TRANS_QUERY_NMPIPE_INFO	0x0022
TRANS_PEEK_NMPIPE	0x0023
TRANS_TRANSACT_NMPIPE	0x0026
TRANS_RAW_WRITE_NMPIPE	0x0031
TRANS_READ_NMPIPE	0x0036
TRANS_WRITE_NMPIPE	0x0037
TRANS_WAIT_NMPIPE	0x0053
TRANS_CALL_NMPIPE	0x0054

Client requests for named pipe operations that require sending more data than the negotiated maximum server buffer size (the **MaxBufferSize** field in the SMB_COM_NEGOTIATE response (section 2.2.2)) MUST be split into multiple requests. The initial primary request MUST be sent as an SMB_COM_TRANSACTION primary client request. The initial primary request MUST include the **SetupCount** bytes and the **Name** field (if specified) for a transaction to be successful. The subsequent secondary requests MUST be sent as multiple SMB_COM_TRANSACTION_SECONDARY secondary client requests. The following subcommand requests MAY require sending more data than the negotiated server maximum buffer size: TRANS_TRANSACT_NMPIPE, TRANS_RAW_WRITE_NMPIPE, TRANS_WRITE_NMPIPE, TRANS_CALL_NMPIPE, and TRANS_MAILSLOT_WRITE.

Server responses that require sending more data than the negotiated maximum client buffer size (the **MaxBufferSize** field in the SMB_COM_SESSION_SETUP request) MUST be sent as multiple SMB_COM_TRANSACTION interim server responses followed by a single final SMB_COM_TRANSACTION server response. The following subcommand server responses MAY require sending more data than the negotiated maximum client buffer size: TRANS_RAW_READ_NMPIPE, TRANS_PEEK_NMPIPE, TRANS_TRANSACT_NMPIPE, TRANS_READ_NMPIPE, and TRANS_CALL_NMPIPE.

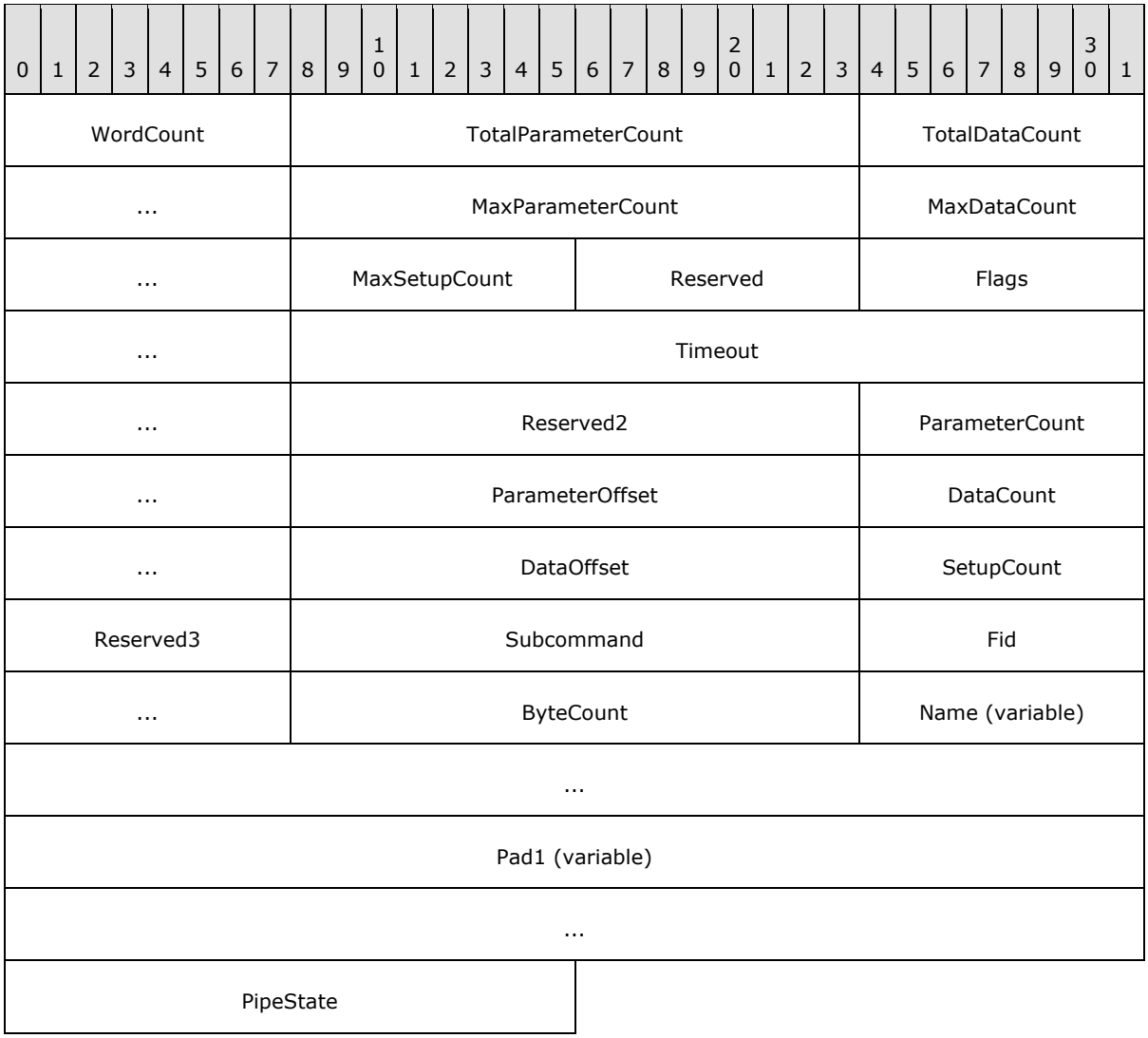
Mailslot operations MUST have a **SetupCount** equal to 3 and a **Name** for the transaction beginning with \MAILSLOT. The first WORD of the setup information MUST be a subcommand code from the list below. If the server receives a request with a subcommand code not listed below, it MUST fail the request with STATUS_NOT_SUPPORTED.<53>

Mailslot subcommand codes	Value
TRANS_MAILSLOT_WRITE	0x0001

2.2.12.1 TRANS_SET_NMPIPE_STATE Request

The TRANS_SET_NMPIPE_STATE subcommand of the SMB_COM_TRANSACTION MUST be used to set the read mode and **blocking mode** of a specified named pipe.

The format follows the convention for SMB_COM_TRANSACTION, as specified in [CIFS] section 3.13.1.



WordCount (1 byte): The count of 16-bit words in the request structure. For this request, the count MUST be 0x10 (16), which is 14 plus the **SetupCount** value of 2.

TotalParameterCount (2 bytes): The number of bytes in the Parameter buffer. For this request, the count MUST be 2. The content of the Parameter buffer is described below.

TotalDataCount (2 bytes): The number of bytes in the Data buffer. For this request, the count MUST be set to 0.

MaxParameterCount (2 bytes): The maximum number of bytes that the client accepts in the Parameter buffer of the response. For this request, the count MUST be set to 0.

MaxDataCount (2 bytes): The maximum number of bytes that the client accepts in the Data buffer of the response. For this request, the count MUST be set to 0.

MaxSetupCount (1 byte): The maximum number of bytes that the client accepts in the Setup buffer of the response. For this request, the count MUST be set to 0.

Reserved (1 byte): An unused value that SHOULD be set to 0 when sending this message. The server MUST ignore this field when receiving this message.

Flags (2 bytes): A set of options that provide additional information to the server on this request. Unused bit fields SHOULD be set to 0 by the client when sending a request, and MUST be ignored when received by the server. The client MAY set either of the following bits.

Value	Meaning
DISCONNECT_TID 0x0001	The server MUST disconnect the tree connect associated with the **Tid** field received in the SMB header of this request after the request completes. The client SHOULD NOT send an SMB_COM_TREE_DISCONNECT for this tree connect.
NO_RESPONSE 0x0002	The server MUST process this client request as a one-way transaction and MUST NOT send a response back to the client.<54>

Timeout (4 bytes): The maximum amount of time in milliseconds to wait for the operation to complete. The client MAY set this to 0 to indicate that no time-out is given. If the operation does not complete within the specified time, the server MAY abort the request and send a failure response.<55>

Reserved2 (2 bytes): An unused value that SHOULD be set to 0 when sending this message. The server MUST ignore this field when receiving this message.

ParameterCount (2 bytes): The count of bytes in the Parameter buffer of this packet. For this request, the count MUST be 2.

ParameterOffset (2 bytes): The offset in bytes from the start of the SMB header for this request to the start of the Parameter buffer. The Parameter buffer MUST start after the **Name** field. The Parameter buffer MUST be aligned to start on a four-byte boundary from the start of the SMB header.

DataCount (2 bytes): This value MUST be the count of bytes in the Data buffer field. For this request, the count MUST be set to 0.

DataOffset (2 bytes): This value MUST be the offset in bytes to the start of the Data buffer field from the start of the SMB header for this request. This value MUST be greater than or equal to **ParameterOffset** plus **ParameterCount**. The Data buffer MUST be aligned to start on a four-byte boundary from the start of the SMB header. If the **DataCount** field is 0, the client MAY set the **DataOffset** field to 0, or the client MAY set this field to a location in the packet after the point where the Parameter bytes would be placed, if present.<56>

SetupCount (1 byte): This value MUST be the number of setup words. For this request, the count MUST be set to 2.

Reserved3 (1 byte): An unused value that SHOULD be set to 0 when sending this message. The server MUST ignore this field when receiving this message.

Subcommand (2 bytes): The first setup word for this request. This field MUST be set to TRANS_SET_NMPIPE_STATE, the subcommand to request. For a list of named pipe subcommands and values, see section 2.2.12.

Fid (2 bytes): The second setup word for this request. This field MUST be the SMB file identifier for the named pipe that is having its state changed. This field MUST be set to a valid Fid from a server response for a previous SMB command to open or create a named pipe. These commands include: SMB_COM_OPEN, SMB_COM_CREATE, SMB_COM_CREATE_TEMPORARY, SMB_COM_CREATE_NEW, SMB_COM_OPEN_ANDX, SMB_COM_NT_CREATE_ANDX, and SMB_COM_NT_TRANSACT with subcommand NT_TRANSACT_CREATE.

ByteCount (2 bytes): This value MUST be the count of bytes that follows the **ByteCount** field in this packet.

Name (variable): The optional name for this transaction. If SMB_FLAGS2_UNICODE is set in the **Flags2** field of the SMB header of the request, the name string MUST be a null-terminated array of 16-bit Unicode characters. Otherwise, the name string MUST be a null-terminated array of OEM characters. A client MAY set this to the named pipe name. The server MUST ignore this field in the request. If the name string consists of Unicode characters, this field MUST be aligned to start on a two-byte boundary from the start of the SMB header.<57>

Pad1 (variable): An optional padding to set the Parameter buffer (the **PipeState** field) to start on a four-byte boundary from the start of the SMB header.

PipeState (2 bytes): The value that describes the state being set on the pipe, as defined below. Any combination of the following flags MUST be valid for the set operation, and all other flags are considered unused and SHOULD be set to 0 when this message is sent. The server MUST ignore these bits when the message is received.

Value	Meaning
Blocking 0x8000	If set, the pipe returns immediately on a read request if no data is available. If not set, a read request blocks until there is data to be read.
ReadMode 0x0100	If set to 1, the named pipe operates in message mode. If set to 0, the named pipe operates in stream mode. In message mode, the system treats the bytes read or written in each I/O operation to the pipe as a message unit. The system always performs write operations on message-type pipes as if write-through mode were enabled. In stream mode, data is read or written to the pipe as a stream of bytes, and the system does not differentiate between the bytes read or written in different I/O operations.

2.2.12.2 TRANS_SET_NMPIPE_STATE Response

A server MUST send a TRANS_SET_NMPIPE_STATE response in reply to a client TRANS_SET_NMPIPE_STATE subcommand request when the request is successful. The server MUST set an error code in the **Status** field of the SMB header of the response to indicate whether the request to set the read mode and blocking mode of a specified named pipe is successful.

The response format follows the convention for SMB_COM_TRANSACTION, as specified in [CIFS] section 3.13.1.

0	1	2	3	4	5	6	7	8	9	10	1	2	3	4	5	6	7	8	9	20	1	2	3	4	5	6	7	8	9	30	1
WordCount								TotalParameterCount																TotalDataCount							
...								Reserved																ParameterCount							
...								ParameterOffset																ParameterDisplacement							
...								DataCount																DataOffset							
...								DataDisplacement																SetupCount							
Reserved2								ByteCount																Pad (optional)							

WordCount (1 byte): A count of 16-bit words in the response structure. For this response, this MUST be 0x0A (10), which is 10 plus the **SetupCount** of 0.

TotalParameterCount (2 bytes): The number of bytes in the Parameter buffer. For this response, it MUST be 0.

TotalDataCount (2 bytes): The number of bytes in the Data buffer. For this response, it MUST be set to 0.

Reserved (2 bytes): An unused value that SHOULD be set to 0 when sending this response. The client MUST ignore this field when receiving this message.

ParameterCount (2 bytes): The number of bytes of the Parameter buffer contained in this packet. For this response, it MUST be set to 0.

ParameterOffset (2 bytes): This value MUST be the offset in bytes from the start of the SMB header for this response to the start of the part of the Parameter buffer contained in this response. The Parameter buffer MUST start after the **Unused2** field. The Parameter buffer MUST be aligned to start on a 4-byte boundary from the start of the SMB header. If the **ParameterCount** field is 0, the server MAY set the **ParameterOffset** field to 0, or the server MAY set this field to a location in the packet after the point where the Parameter bytes would have been placed, if present.<58>

ParameterDisplacement (2 bytes): This field MUST be the offset in bytes into the final Parameter buffer assembled from all responses. For a single buffer transaction (whose Parameter buffer fits in a single response), this value MUST be set to 0.

DataCount (2 bytes): The number of bytes in the Data buffer contained in this packet. For this response, it MUST be set to 0.

DataOffset (2 bytes): This field MUST be the offset in bytes from the start of the SMB header for this response to the start of the Data buffer field. This value MUST be greater than or equal to **ParameterOffset** plus **ParameterCount**. The Data buffer MUST be aligned to start on a 4-byte boundary from the start of the SMB header. If the **DataCount** field is 0, the server MAY set the **DataOffset** field to 0, or the server MAY set this field to a location in the packet after the point where the Parameter bytes would be placed, if present.<59>

DataDisplacement (2 bytes): This value MUST be the offset in bytes into the final Data buffer assembled from all responses. For a single buffer transaction (whose Data buffer fits in a single response), this value MUST be set to 0.

SetupCount (1 byte): The number of setup words. For this response, it MUST be set to 0.

Reserved2 (1 byte): An unused value that SHOULD be set to 0 when sending this response. The client MUST ignore this field when receiving this message.

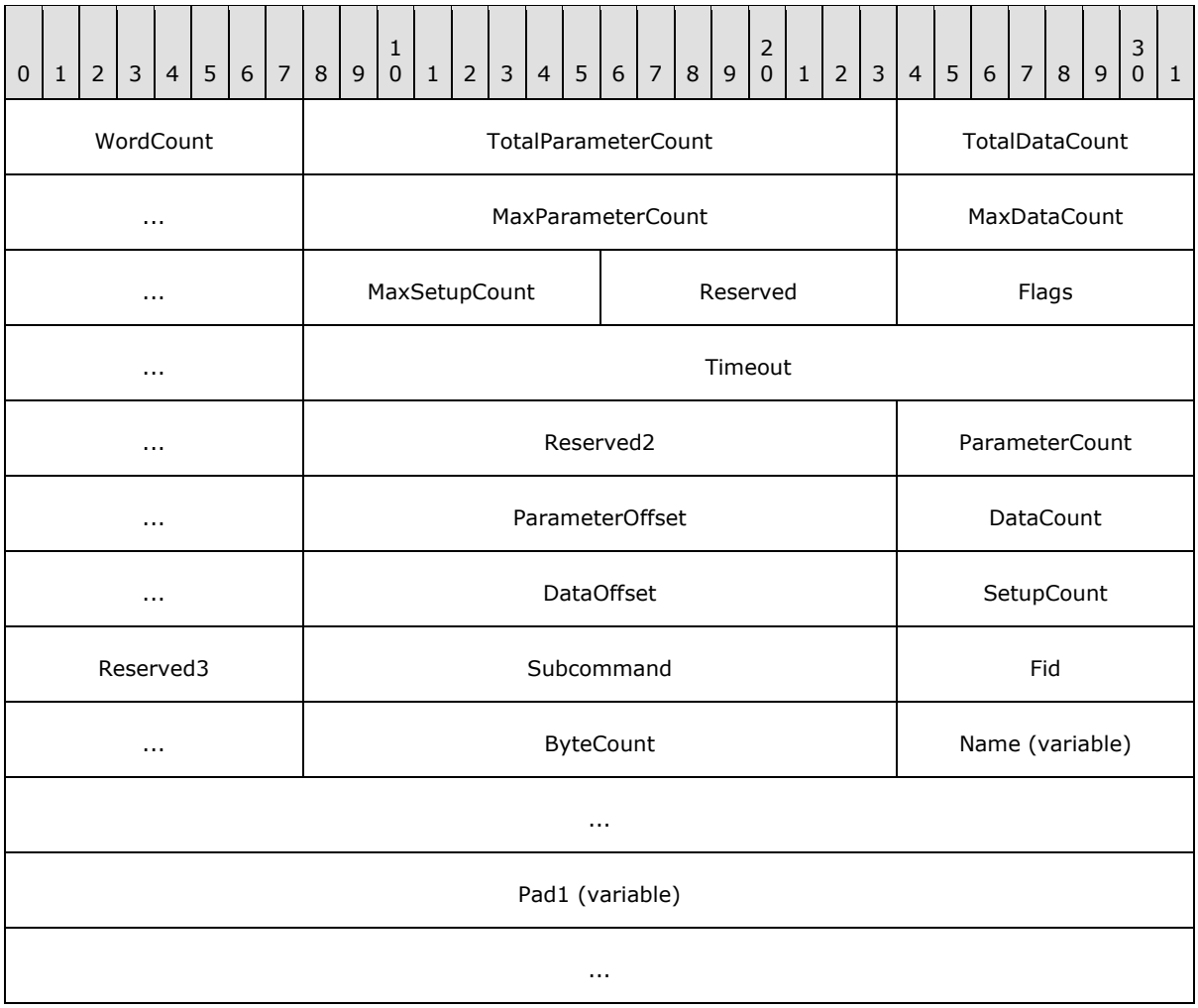
ByteCount (2 bytes): This value MUST be the number of bytes that follows the **ByteCount** field.

Pad (1 byte): Potential padding required to set the Parameter buffer that MUST BE on a 4-byte boundary.

2.2.12.3 TRANS_RAW_READ_NMPIPE Request

The TRANS_RAW_READ_NMPIPE subcommand of the SMB_COM_TRANSACTION allows for a **raw read** of data from a named pipe. This method of reading data from a named pipe ignores message boundaries even if the pipe was set up as a message mode pipe.<60>

The format follows the convention for SMB_COM_TRANSACTION, as specified in [CIFS] section 3.13.1.



WordCount (1 byte): A count of 16-bit words in the request structure. For this request, it MUST be 0x10 (16), which is 14 plus the **SetupCount** of 2.

TotalParameterCount (2 bytes): The number of bytes in the Parameter buffer. For this request, it MUST be set to 0.

TotalDataCount (2 bytes): The number of bytes in the Data buffer. For this request, it MUST be set to 0.

MaxParameterCount (2 bytes): The maximum number of bytes the client will accept in the Parameter buffer of the response. For this request, it MUST be set to 0.

MaxDataCount (2 bytes): This value MUST be the number of bytes the client requests to read from the named pipe.

MaxSetupCount (1 byte): The maximum number of bytes the client accepts in the Setup buffer of the response. For this request, it MUST be set to 0.

Reserved (1 byte): An unused value that SHOULD be set to 0 when sending this request. The server MUST ignore this field when receiving this message.

Flags (2 bytes): A set of options that provide additional information to the server on this request. The client MAY set either of the following bits. Unused bit fields SHOULD be set to 0 by the client and MUST be ignored when received by the server.

Value	Meaning
DISCONNECT_TID 0x0001	The server MUST disconnect the tree connect associated with the **Tid** field received in the SMB header of this request after the request completes. The client SHOULD NOT send an SMB_COM_TREE_DISCONNECT for this tree connect.
NO_RESPONSE 0x0002	The server MUST process this client request as a one-way transaction and MUST NOT send a response back to the client.<61>

Timeout (4 bytes): This value MUST be the maximum amount of time in milliseconds to wait for the operation to complete. The client MAY set this to 0 to indicate that no time-out is given. If the operation does not complete within the specified time, the server MAY abort the request and send a failure response.<62>

Reserved2 (2 bytes): An unused value that SHOULD be set to 0 when sending this request. The server MUST ignore this field when receiving this message.

ParameterCount (2 bytes): The count of bytes in the Parameter buffer of this packet. For this request, it MUST be set to 0.

ParameterOffset (2 bytes): This value MUST be the offset in bytes from the start of the SMB header for this request to the start of the Parameter buffer. The Parameter buffer MUST start after the **Name** field. The Parameter buffer MUST be aligned to start on a 4-byte boundary from the start of the SMB header. If the **ParameterCount** field is 0, the client MAY set the **ParameterOffset** field to 0, or the client MAY set this field to a location in the packet after the **Name** field where Parameter bytes would have been placed, if present.<63>

DataCount (2 bytes): The count of bytes in the Data buffer field. For this request, it MUST be set to 0.

DataOffset (2 bytes): An offset to the Data buffer field in this packet. It MUST be set to the offset of the Data buffer field from the start of the SMB header for this request. This value MUST be greater than or equal to **ParameterOffset** plus **ParameterCount**, and MUST be aligned on a 4-byte boundary. If the **DataCount** field is 0, the client MAY set the **DataOffset** field to 0, or the client MAY set this field to a location in the packet after the point where the Parameter bytes would be placed, if present.<64>

SetupCount (1 byte): The number of setup words. For this request, it MUST be 2.

Reserved3 (1 byte): An unused value that SHOULD be set to 0 when sending this request. The server MUST ignore this field when receiving this message.

Subcommand (2 bytes): The first setup word for this request. This field MUST be set to TRANS_RAW_READ_NMPIPE, the subcommand to request. For a list of named pipe subcommands and values, see section 2.2.12.

Fid (2 bytes): The second setup word for this request. This field is the SMB file identifier for the named pipe to read. This field MUST be set to a valid Fid from a server response for a previous SMB command to open or create a named pipe. These commands include SMB_COM_OPEN, SMB_COM_CREATE, SMB_COM_CREATE_TEMPORARY, SMB_COM_CREATE_NEW, SMB_COM_OPEN_ANDX, SMB_COM_NT_CREATE_ANDX, and SMB_COM_NT_TRANSACT with subcommand NT_TRANSACT_CREATE.

ByteCount (2 bytes): This value MUST be the count of bytes that follows the **ByteCount** field in this packet.

Name (variable): An optional name for this transaction. If SMB_FLAGS2_UNICODE is set in the **Flags2** field of the SMB header of the request, the name string MUST be a null-terminated array of 16-bit Unicode characters. Otherwise, the name string MUST be a null-terminated array of OEM characters. A client MAY set this to the named pipe name. The server MUST ignore this field in the request. If the name string consists of Unicode characters, this field MUST be aligned to start on a 2-byte boundary from the start of the SMB header.<65>

Pad1 (variable): An optional padding to set the start of the Parameter buffer that MUST be set on a 4-byte boundary from the start of the SMB header.

2.2.12.4 TRANS_RAW_READ_NMPIPE Response

This message MUST be sent by a server to respond to a client TRANS_RAW_READ_NMPIPE request when the request is successful. The server MUST set an error code in the **Status** field of the SMB header of the response to indicate whether the read from the named pipe was successful or failed.

The format follows the convention for SMB_COM_TRANSACTION, as specified in [CIFS] section 3.13.1.

0	1	2	3	4	5	6	7	8	9	10	1	2	3	4	5	6	7	8	9	20	1	2	3	4	5	6	7	8	9	30	1
WordCount								TotalParameterCount																TotalDataCount							
...								Reserved																ParameterCount							
...								ParameterOffset																ParameterDisplacement							
...								DataCount																DataOffset							
...								DataDisplacement																SetupCount							
Reserved2								ByteCount																Pad1 (optional)							
Data Buffer (variable)																															
...																															

WordCount (1 byte): The count of 16-bit words in the response structure. For this response, this MUST be 0x0A (10), which is 10 plus the **SetupCount** of 0.

TotalParameterCount (2 bytes): The number of bytes in the Parameter buffer. For this response, it MUST be set to 0.

TotalDataCount (2 bytes): This value MUST be the number of bytes read from the named pipe in raw format.

Reserved (2 bytes): An unused value that SHOULD be set to 0 when sending this response. The client MUST ignore this field when receiving this message.

ParameterCount (2 bytes): The number of bytes of the Parameter buffer contained in this packet. For this response, it MUST be set to 0.

ParameterOffset (2 bytes): An offset in bytes from the start of the SMB header for this response to the start of the part of the Parameter buffer contained in this response. The Parameter buffer MUST start after the **Unused2** field. The Parameter buffer MUST be aligned to start on a 4-byte boundary from the start of the SMB header. If the **ParameterCount** field is 0, the server MAY set the **ParameterOffset** field to 0, or the server MAY set this field to a location in the packet after the point where Parameter bytes would have been placed, if present.<66>

ParameterDisplacement (2 bytes): An offset in bytes into the final Parameter buffer assembled from all responses. For a single buffer transaction (whose Parameter buffer fits in a single response), this value MUST be set to 0.

DataCount (2 bytes): The number of bytes in the Data buffer contained in this packet. For this response, it MUST be set to less than or equal to the value of the **TotalDataCount** field.

DataOffset (2 bytes): An offset to the Data buffer field in this packet. It MUST be set to the offset of the Data buffer field from the start of the SMB header for this response. This value MUST be greater than or equal to **ParameterOffset** plus **ParameterCount**. The Data buffer MUST be aligned to start on a 4-byte boundary from the start of the SMB header. If the **DataCount** field is 0, the server MAY set the **DataOffset** field to 0, or the server MAY set this field to a location in the packet after the point where the Parameter bytes would be placed, if present.<67>

DataDisplacement (2 bytes): An offset in bytes into the final Data buffer assembled from all responses. For a single buffer transaction (whose Data buffer fits in a single response), this value MUST be set to 0.

SetupCount (1 byte): The number of setup words. For this response, it MUST be set to 0.

Reserved2 (1 byte): An unused value that SHOULD be set to 0 when sending this response. The client MUST ignore this field when receiving this message.

ByteCount (2 bytes): This value MUST be the count of variable-length bytes that follows the **ByteCount** field.

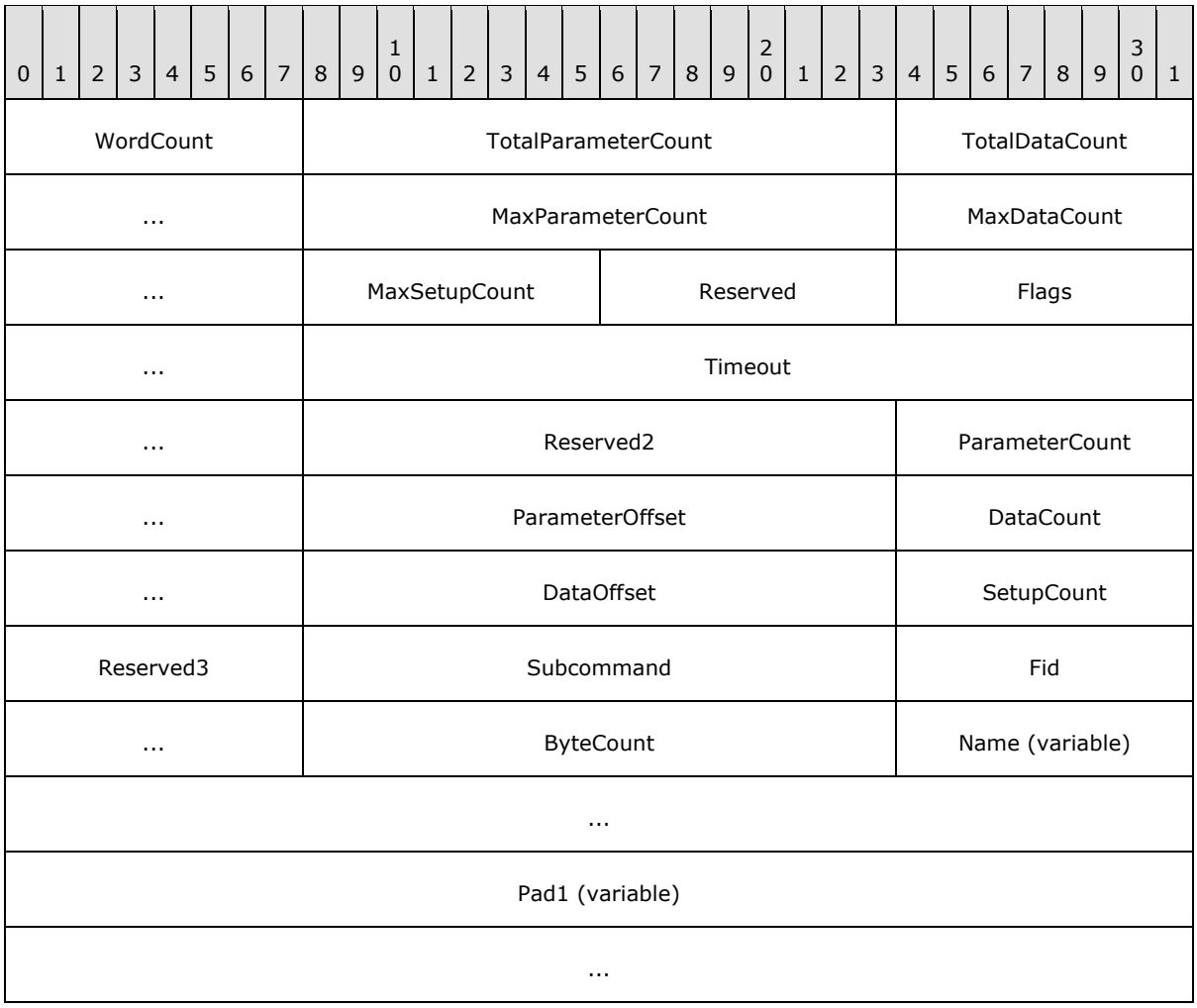
Pad1 (1 byte): This value MUST be the potential padding required to set the start of the Data buffer on a 4-byte boundary from the start of the SMB header.

Data Buffer (variable): The Data buffer that MUST contain the bytes read from the named pipe in raw mode. The size of the buffer MUST be equal to the value in **DataCount**.

2.2.12.5 TRANS_QUERY_NMPIPE_STATE Request

The TRANS_QUERY_NMPIPE_STATE subcommand of the SMB_COM_TRANSACTION allows for a client to retrieve information about a specified named pipe.

The format follows the convention for SMB_COM_TRANSACTION, as specified in [CIFS] section 3.13.1.



WordCount (1 byte): The count of 16-bit words in the request structure. For this request, it MUST be 0x10 (16), which is 14 plus the **SetupCount** of 2.

TotalParameterCount (2 bytes): The number of bytes in the Parameter buffer. For this request, it MUST be set to 0.

TotalDataCount (2 bytes): The number of bytes in the Data buffer. For this request, it MUST be set to 0.

MaxParameterCount (2 bytes): The maximum number of bytes the client will accept in the Parameter buffer of the response. For this request, it MUST be set to 2.

MaxDataCount (2 bytes): The maximum number of bytes the client will accept in the Data buffer of the response. For this request, it MUST be set to 0.

MaxSetupCount (1 byte): The maximum number of bytes the client will accept in the Setup buffer of the response. For this request, it MUST be set to 0.

Reserved (1 byte): An unused value that SHOULD be set to 0 when sending this request. The server MUST ignore this field when receiving this message.

Flags (2 bytes): A set of options that provides additional information to the server on this request. The client MAY set either of the following bits. Unused bit fields SHOULD be set to 0 by the client when sending a response, and MUST be ignored when received by the client.

Value	Meaning
DISCONNECT_TID 0x0001	Server MUST disconnect the tree connect associated with the Tid received in the SMB header of this request after the request completes. The client SHOULD NOT send an SMB_COM_TREE_DISCONNECT for this tree connect.
NO_RESPONSE 0x0002	The server MUST process this client request as a one-way transaction and MUST NOT send a response back to the client.<68>

Timeout (4 bytes): The maximum amount of time in milliseconds to wait for the operation to complete. The client MAY set this to 0 to indicate that no time-out is given. If the operation does not complete within the specified time, the server MAY abort the request and send a failure response.<69>

Reserved2 (2 bytes): An unused value that SHOULD be set to 0 when sending this request. The server MUST ignore this field when receiving the message.

ParameterCount (2 bytes): The count of bytes in the Parameter buffer of this packet. For this request, it MUST be set to 0.

ParameterOffset (2 bytes): An offset in bytes from the start of the SMB header for this response to where the Parameter buffer begins. The Parameter buffer MUST start after the **Name** field. The Parameters buffer MUST be aligned on a four-byte boundary. If the **ParameterCount** field is 0, the client MAY set the **ParameterOffset** field to 0, or the client MAY set this field to a location in the packet after the **Name** field where Parameter bytes would have been placed, if present.<70>

DataCount (2 bytes): The count of bytes in the Data buffer field. For this request, it MUST be set to 0.

DataOffset (2 bytes): An offset to the Data buffer field in this packet. It MUST be set to the offset of the Data buffer field from the start of the SMB header for this request. The Data buffer MUST be greater than or equal to **ParameterOffset** plus **ParameterCount**. The data buffer MUST be aligned on a four-byte boundary. If the **DataCount** field is 0, the client MAY set the **DataOffset** field to 0, or the client MAY set this field to a location in the packet after where the Parameter bytes would be placed, if present.<71>

SetupCount (1 byte): The number of setup words. For this request, it MUST be 2.

Reserved3 (1 byte): A reserved value that SHOULD be set to 0 when sending this request. The server MUST ignore this field when receiving this message.

Subcommand (2 bytes): First setup word for this request. This field MUST be set to TRANS_QUERY_NMPIPE_STATE, the subcommand to request. For a list of named pipe subcommands and values, see section 2.2.12.

Fid (2 bytes): Second setup word for this request. This field is the SMB file identifier for the named pipe that is being queried. This field MUST be set to a valid Fid from a server response for a previous SMB command to open or create a named pipe. These commands include SMB_COM_OPEN, SMB_COM_CREATE, SMB_COM_CREATE_TEMPORARY, SMB_COM_CREATE_NEW, SMB_COM_OPEN_ANDX, SMB_COM_NT_CREATE_ANDX, and SMB_COM_NT_TRANSACT with subcommand NT_TRANSACT_CREATE.

ByteCount (2 bytes): This value MUST be set to the number of bytes that follows the **ByteCount** field in the packet.

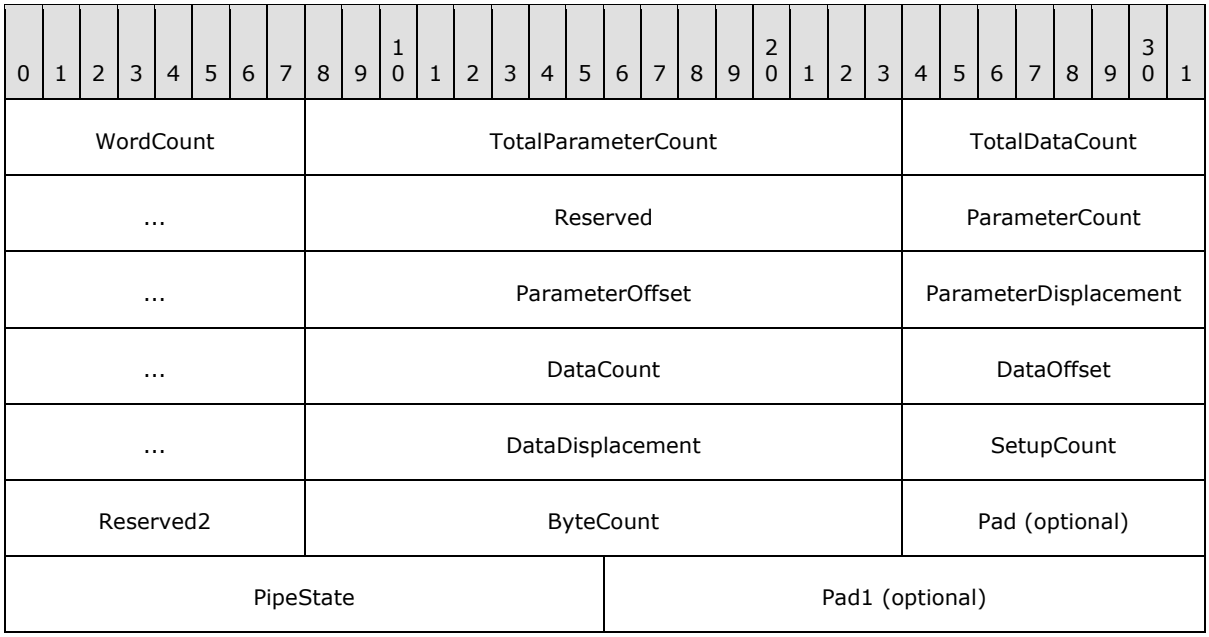
Name (variable): An optional name for this transaction. If SMB_FLAGS2_UNICODE is set in the **Flags2** field of the SMB header of the request, the name string MUST be a null-terminated array of 16-bit Unicode characters. Otherwise, the name string MUST be a null-terminated array of OEM characters. A client MAY set this to the named pipe name. The server MUST ignore this field in the request. If the name string consists of Unicode characters, this field MUST be aligned to start on a two-byte boundary from the start of the SMB header.<72>

Pad1 (variable): An optional padding to set the start of the Parameter buffer that MUST be on a 4-byte boundary from the start of the SMB header.

2.2.12.6 TRANS_QUERY_NMPIPE_STATE Response

The server MUST send a TRANS_QUERY_NMPIPE_STATE response in reply to a client TRANS_RAW_READ_NMPIPE subcommand request when the request is successful. The server MUST set an error code in the **Status** field of the SMB header of the response to indicate whether the query operation on the named pipe was successful or failed.

The format follows the convention for SMB_COM_TRANSACTION, as specified in [CIFS] section 3.13.1.



WordCount (1 byte): The count of 16-bit words in the response structure. For this response, this MUST be 0x0A (10), which is 10 plus the **SetupCount** of 0.

TotalParameterCount (2 bytes): The number of bytes in the Parameter buffer. For this response, it MUST be set to 2.

TotalDataCount (2 bytes): The number of bytes in the Data buffer. For this response, it MUST be set to 0.

Reserved (2 bytes): An unused value that SHOULD be set to 0 when sending this response. The client MUST ignore this field when receiving this message.

ParameterCount (2 bytes): The number of bytes of the Parameter buffer contained in this packet. For this response, it MUST be set to 2 (the size of the **PipeState** field).

ParameterOffset (2 bytes): An offset in bytes from the start of the SMB header for this response to the start of the part of the Parameter buffer contained in this response. The Parameter buffer MUST start after the **Unused2** field. The Parameter buffer MUST be aligned to start on a 4-byte boundary from the start of the SMB header.

ParameterDisplacement (2 bytes): An offset in bytes into the final Parameter buffer assembled from all responses. For a single buffer transaction (whose Parameter buffer fits in a single response), this value MUST be set to 0.

DataCount (2 bytes): The number of bytes in the Data buffer contained in this packet. For this response, it MUST be set to 0.

DataOffset (2 bytes): An offset in bytes to the start of the Data buffer field from the start of the SMB header in this response. The Data buffer MUST be greater than or equal to **ParameterOffset** plus **ParameterCount**. The data buffer MUST be aligned to start on a 4-byte boundary from the start of the SMB header. If the **DataCount** field is 0, the server MAY set the **DataOffset** field to 0, or the server MAY set this field to a location in the packet after where the Parameter bytes would be placed, if present.<73>

DataDisplacement (2 bytes): An offset in bytes into the final Data buffer assembled from all responses. For a single buffer transaction (whose Data buffer fits in a single response), this value MUST be set to 0.

SetupCount (1 byte): The number of setup words. For this response, it MUST be set to 0.

Reserved2 (1 byte): An unused value that SHOULD be set to 0 when sending this response. The client MUST ignore this field when receiving this message.

ByteCount (2 bytes): This value MUST be the number of bytes that follows the **ByteCount** field.

Pad (1 byte): The potential padding required to set the start of the Parameter buffer that MUST be on a 4-byte boundary from the start of the SMB header.

PipeState (2 bytes): The Parameter buffer contains the **PipeState** field, and MUST indicate the state of the named pipe. Unused bit fields SHOULD be set to 0 by the server when sending a response, and MUST be ignored when received by the client.

Value	Meaning
ICount 0x00FF	An 8-bit count to control pipe instancing.
ReadMode 0x0100	This bit field indicates the client read mode for the named pipe. If set to 1, it indicates that the client opened the named pipe or set the named pipe to message mode. If set to 0, the named pipe was opened or set to byte stream mode. This bit field has no effect on writes to the named pipe.
NamedPipeType 0x0400	This bit field indicates the type of the named pipe when the named pipe was created by the server. If set to 1, the named pipe was created by the server as a message mode pipe. If set to 0, the named pipe was created as a stream mode

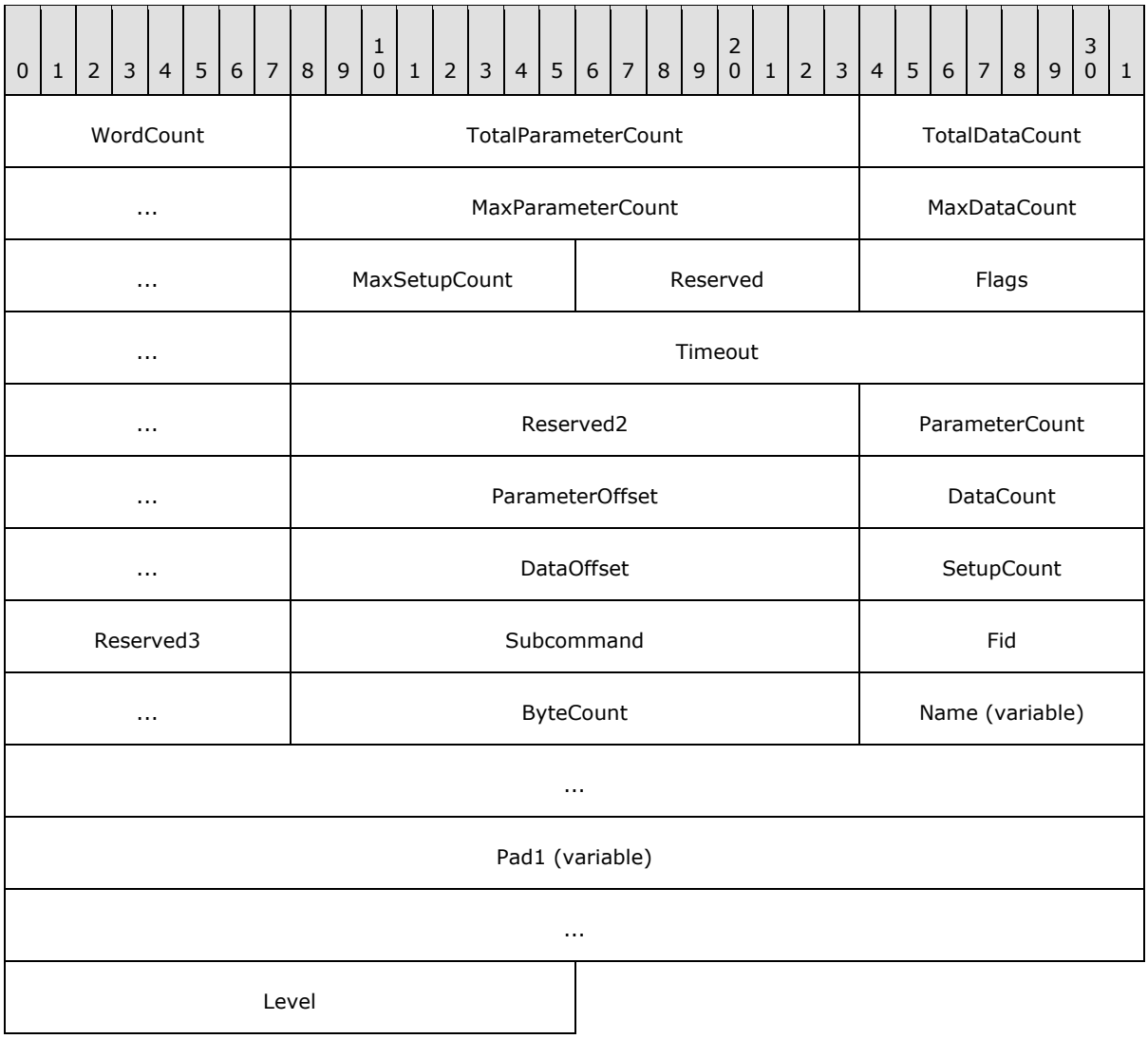
Value	Meaning
	pipe.
EndPoint 0x4000	A value that indicates the endpoint of the pipe. A value of 0 indicates the client end of the pipe. A value of 1 indicates the server end of the pipe. The SMB server MUST set the EndPoint bit to 0 when responding to the client request because the endpoint specified is the client end of the pipe.
Blocking 0x8000	If set, indicates that named pipe read requests return immediately if no data is available, and writes return immediately. If not set, named pipe read requests block if no data is available, and writes block until their data is consumed by a read.

Pad1 (2 bytes): The potential padding required to set the start of the Parameter buffer that MUST be on a 4-byte boundary from the start of the SMB header.

2.2.12.7 TRANS_QUERY_NMPIPE_INFO Request

The TRANS_QUERY_NMPIPE_INFO subcommand of the SMB_COM_TRANSACTION is used to retrieve pipe information about a named pipe.

The format follows the convention for SMB_COM_TRANSACTION, as specified in [CIFS] section 3.13.1.



WordCount (1 byte): The count of 16-bit words in the request structure. For this request, it MUST be 0x10 (16), which is 14 plus the **SetupCount** of 2.

TotalParameterCount (2 bytes): The number of bytes in the Parameter buffer. For this request, it MUST be 2.

TotalDataCount (2 bytes): The number of bytes in the Data buffer. For this request, it MUST be set to 0.

MaxParameterCount (2 bytes): The maximum number of bytes the client will accept in the Parameter buffer of the response. For this request, it MUST be set to 0.

MaxDataCount (2 bytes): The maximum number of bytes that the client will accept in the Data buffer of the response. For this request, it MUST be greater than or equal to 0x40 (64).

MaxSetupCount (1 byte): The maximum number of bytes the client will accept in the Setup buffer of the response. For this request, it MUST be set to 0.

Reserved (1 byte): An unused value that SHOULD be set to 0 when sending this request. The server MUST ignore this field when receiving this message.

Flags (2 bytes): A set of options that provide additional information to the server on this request. The client MAY set either of the following bits. Unused bit fields SHOULD be set to 0 by the client when sending a request, and MUST be ignored when received by the server.

Value	Meaning
DISCONNECT_TID 0x0001	The server MUST disconnect the tree connect associated with the Tid received in the SMB header of this request after the request completes. The client SHOULD NOT send an SMB_COM_TREE_DISCONNECT for this tree connect.
NO_RESPONSE 0x0002	The server MUST process this client request as a one-way transaction and MUST NOT send a response back to the client.<74>

Timeout (4 bytes): The maximum amount of time in milliseconds to wait for the operation to complete. The client MAY set this to 0 to indicate that no time-out is given. If the operation does not complete within the specified time, the server MAY abort the request and send a failure response.<75>

Reserved2 (2 bytes): An unused value that SHOULD be set to 0 when sending this request. The server MUST ignore this field when receiving this message.

ParameterCount (2 bytes): The count of bytes in the Parameter buffer of this packet. For this request, it MUST be set to 2.

ParameterOffset (2 bytes): An offset in bytes from the start of the SMB header for this request to the start of the Parameter buffer. The Parameter buffer MUST start after the **Name** field. The Parameter buffer MUST be aligned to start on a 4-byte boundary from the start of the SMB header.

DataCount (2 bytes): The count of bytes in the Data buffer field. For this request, it MUST be set to 0.

DataOffset (2 bytes): An offset in bytes from the start of the SMB header to the start of the Data buffer field in this packet. The Data buffer MUST be greater than or equal to **ParameterOffset** plus **ParameterCount**. The Data buffer MUST be aligned to start on a 4-byte boundary from the start of the SMB header. If the **DataCount** field is 0, the client MAY set the **DataOffset** field to 0, or the client MAY set this field to a location in the packet after where the Parameter bytes would be placed, if present.<76>

SetupCount (1 byte): The number of setup words. For this request, it MUST be set to 2.

Reserved3 (1 byte): An unused value that SHOULD be set to 0 when sending this request. The server MUST ignore this field when receiving this message.

Subcommand (2 bytes): The first setup word for this request. This field MUST be set to TRANS_QUERY_NMPIPE_INFO, the subcommand to request. For a list of named pipe subcommands and values, see section 2.2.12.

Fid (2 bytes): The second setup word for this request. This field is the SMB file identifier for the named pipe that is being queried. This field MUST be set to a valid Fid from a server response for a previous SMB command to open or create a named pipe. These commands include SMB_COM_OPEN, SMB_COM_CREATE, SMB_COM_CREATE_TEMPORARY, SMB_COM_CREATE_NEW, SMB_COM_OPEN_ANDX, SMB_COM_NT_CREATE_ANDX, and SMB_COM_NT_TRANSACT with subcommand NT_TRANSACT_CREATE.

ByteCount (2 bytes): This value MUST be the number of bytes that follows the **ByteCount** field in this packet.

Name (variable): An optional name for this transaction. If SMB_FLAGS2_UNICODE is set in the **Flags2** field of the SMB header of the request, the name string is a null-terminated array of 16-bit Unicode characters. Otherwise, the name string is a null-terminated array of OEM characters. A client MAY set this to the named pipe name. The server MUST ignore this field in the request. If the name string consists of Unicode characters, this field MUST be aligned to start on a 2-byte boundary from the start of the SMB header.<77>

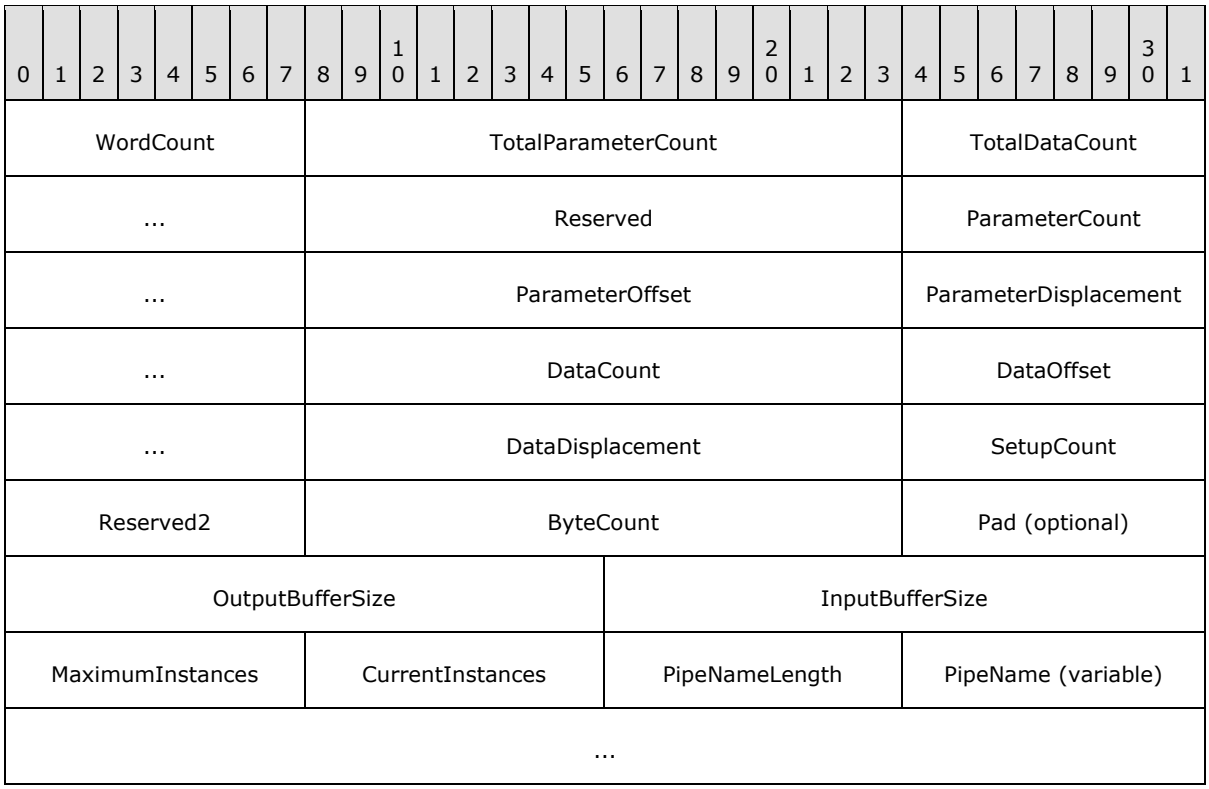
Pad1 (variable): An optional padding to set the start of the Parameter buffer that MUST be on a 4-byte boundary from the start of the SMB header.

Level (2 bytes): A USHORT value (as specified in [MS-DTYP] section **2.2.57**) that describes the information level being queried for the pipe. The only supported value is 0x0001. If the server receives any other value, it MUST fail the request with STATUS_INVALID_PARAMETER.

2.2.12.8 TRANS_QUERY_NMPIPE_INFO Response

The server MUST send a TRANS_QUERY_NMPIPE_INFO response in reply to a client TRANS_QUERY_NMPIPE_INFO subcommand request when the request is successful. The server MUST set an error code in the **Status** field of the SMB header of the response to indicate whether the query operation on the named pipe was successful or failed.

The format follows the convention for SMB_COM_TRANSACTION, as specified in [CIFS] section 3.13.1.



WordCount (1 byte): The count of 16-bit words in the response structure. For this response, this MUST be 0x0A (10), which is 10 plus the **SetupCount** of 0.

TotalParameterCount (2 bytes): The number of bytes in the Parameter buffer. For this response, it MUST be set to 0.

TotalDataCount (2 bytes): The number of bytes in the Data buffer. For this response, it MUST be greater than or equal to 0x40 (64).

Reserved (2 bytes): An unused value that SHOULD be set to 0 when sending this response. The client MUST ignore this field when receiving this message.

ParameterCount (2 bytes): The number of bytes of the Parameter buffer contained in this packet. For this response, it MUST be set to 0.

ParameterOffset (2 bytes): An offset in bytes from the start of the SMB header for this response to the start of the part of the Parameter buffer contained in this response. The Parameter buffer MUST start after the **Unused2** field. The Parameter buffer MUST be aligned to start on a 4-byte boundary from the start of the SMB header. If the **ParameterCount** field is 0, the server MAY set the **ParameterOffset** field to 0, or the server MAY set this field to a location in the packet after where Parameter bytes would have been placed, if present.<78>

ParameterDisplacement (2 bytes): An offset in bytes into the final Parameter buffer assembled from all responses. For a single buffer transaction (whose Parameter buffer fits in a single response), this value MUST be set to 0.

DataCount (2 bytes): The number of bytes in the Data buffer contained in this packet. For this response, it MUST be less than or equal to **TotalDataCount**.

DataOffset (2 bytes): An offset to the Data buffer field in this packet. It MUST be set to the offset of the Data buffer field from the start of the SMB header for this response. The Data buffer MUST be greater than or equal to **ParameterOffset** plus **ParameterCount**. The Data buffer MUST be aligned to start on a 4-byte boundary from the start of the SMB header.

DataDisplacement (2 bytes): An offset in bytes into the final Data buffer assembled from all responses. For a single buffer transaction (whose Data buffer fits in a single response), this value MUST be set to 0.

SetupCount (1 byte): The number of setup words. For this response, it MUST be set to 0.

Reserved2 (1 byte): An unused value that SHOULD be set to 0 when sending this response. The client MUST ignore this field when receiving this message.

ByteCount (2 bytes): This value MUST be the count of bytes that follows the **ByteCount** field in this packet.

Pad (1 byte): The potential padding required to set the start of the Parameter buffer that MUST be on a 4-byte boundary from the start of the SMB header.

OutputBufferSize (2 bytes): The first word of the data buffer that MUST contain the actual size of the buffer for outgoing (server) I/O.

InputBufferSize (2 bytes): This value MUST be the actual size of the buffer for incoming (client) I/O.

MaximumInstances (1 byte): This value MUST be the maximum allowed number of named **pipe instances**.

CurrentInstances (1 byte): This value MUST be the current number of named pipe instances.

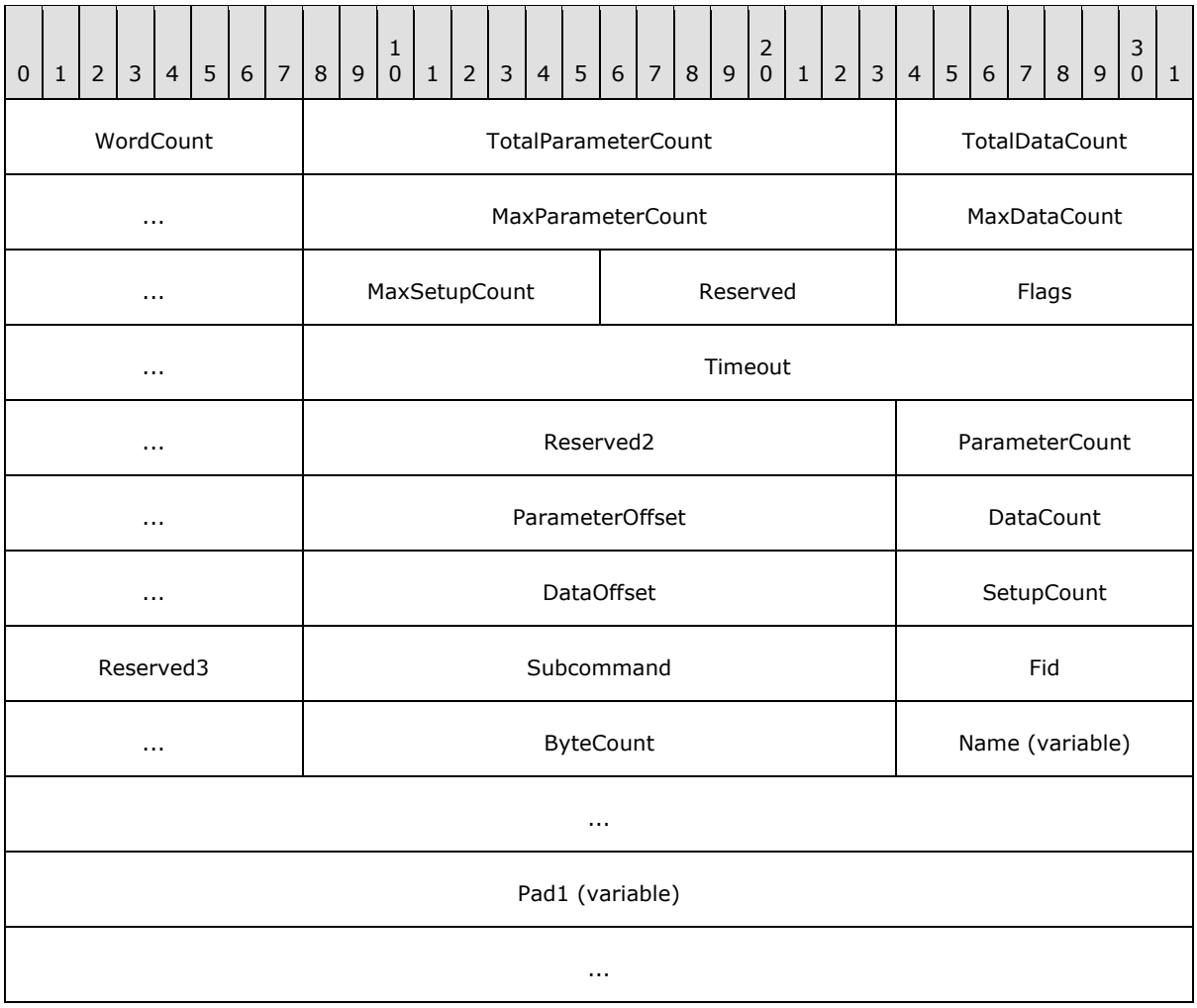
PipeNameLength (1 byte): This value MUST be the length, in bytes, of the pipe name, including the terminating null character.

PipeName (variable): This value MUST be a null-terminated string containing the name of the named pipe, not including the initial \\NodeName string. (that is, of the form \PIPE\pipename). If SMB_FLAGS2_UNICODE is set in the **Flags2** field of the SMB header of the request, the name string MUST be in a null-terminated array of 16-bit Unicode characters. Otherwise, the name string MUST be a null-terminated array of OEM characters. If the **PipeName** field consists of Unicode characters, this field MUST be aligned to start on a 2-byte boundary from the start of the SMB header.

2.2.12.9 TRANS_PEEK_NMPIPE Request

This TRANS_PEEK_NMPIPE subcommand of the SMB_COM_TRANSACTION is used to copy data out of a named pipe without removing it from the named pipe.

The format follows the convention for SMB_COM_TRANSACTION, as specified in [CIFS] section 3.13.1.



WordCount (1 byte): The count of 16-bit words in the request structure. For this request, it MUST be 0x10 (16), which is 14 plus the **SetupCount** of 2.

TotalParameterCount (2 bytes): The number of bytes in the Parameter buffer. For this request, it MUST be set to 0. The content of the Parameter buffer is described below.

TotalDataCount (2 bytes): The number of bytes in the Data buffer. For this request, it MUST be set to 0.

MaxParameterCount (2 bytes): The maximum number of bytes the client will accept in the Parameter buffer of the response. For this request, it MUST be set to 6.

MaxDataCount (2 bytes): The maximum number of bytes that the client will accept in the Data buffer of the response. For this request, it MUST be greater than or equal to 0.

MaxSetupCount (1 byte): The maximum number of bytes that the client will accept in the Setup buffer of the response. For this request, it MUST be set to 0.

Reserved (1 byte): An unused value that SHOULD be set to 0 when sending this message. The server MUST ignore this field when receiving this message.

Flags (2 bytes): A set of options that provide additional information to the server on this request. The client MAY set either of the following bits. Unused bit fields SHOULD be set to 0 by the client when sending a request, and MUST be ignored when received by the server.

Value	Meaning
DISCONNECT_TID 0x0001	The server MUST disconnect the tree connect associated with the Tid received in the SMB header of this request after the request completes. The client SHOULD NOT send an SMB_COM_TREE_DISCONNECT for this tree connect.
NO_RESPONSE 0x0002	The server MUST process this client request as a one-way transaction and MUST NOT send a response back to the client.<79>

Timeout (4 bytes): The maximum amount of time in milliseconds to wait for the operation to complete. The client MAY set this to 0 to indicate that no time-out is given. If the operation does not complete within the specified time, the server MAY abort the request and send a failure response.<80>

Reserved2 (2 bytes): An unused value that SHOULD be set to 0 when sending this message. The server MUST ignore this field when receiving this message.

ParameterCount (2 bytes): The count of bytes in the Parameter buffer of this packet. For this request, it MUST be set to 2.

ParameterOffset (2 bytes): An offset in bytes from the start of the SMB header for this request to the start of the Parameter buffer. The Parameter buffer MUST start after the **Name** field. The Parameter buffer MUST be aligned to start on a 4-byte boundary from the start of the SMB header.

DataCount (2 bytes): The count of bytes in the Data buffer field. For this request, it MUST be set to 0.

DataOffset (2 bytes): An offset in bytes to the start of the Data buffer field from the start of the SMB header for this request. This value MUST be greater than or equal to **ParameterOffset** plus **ParameterCount**. The Data buffer MUST be aligned to start on a 4-byte boundary from the start of the SMB header. If the **DataCount** field is 0, the client MAY set the **DataOffset** field to 0, or the client MAY set this field to a location in the packet after where the Parameter bytes would be placed, if present.<81>

SetupCount (1 byte): The number of setup words. For this request, it MUST be set to 2.

Reserved3 (1 byte): An unused value that SHOULD be set to 0 when sending this message. The server MUST ignore this field when receiving this message.

Subcommand (2 bytes): The first setup word for this request. This field MUST be set to TRANS_PEEK_NMPIPE, the subcommand to request. For a list of named pipe subcommands and values, see section 2.2.12.

Fid (2 bytes): The second setup word for this request. This field MUST be the SMB file identifier for the named pipe that is having its state changed. This field MUST be set to a valid Fid from a server response for a previous SMB command to open or create a named pipe. These commands include SMB_COM_OPEN, SMB_COM_CREATE, SMB_COM_CREATE_TEMPORARY, SMB_COM_CREATE_NEW, SMB_COM_OPEN_ANDX, SMB_COM_NT_CREATE_ANDX, and SMB_COM_NT_TRANSACT with subcommand NT_TRANSACT_CREATE.

ByteCount (2 bytes): This value MUST be the number of bytes that follows the **ByteCount** field in this packet.

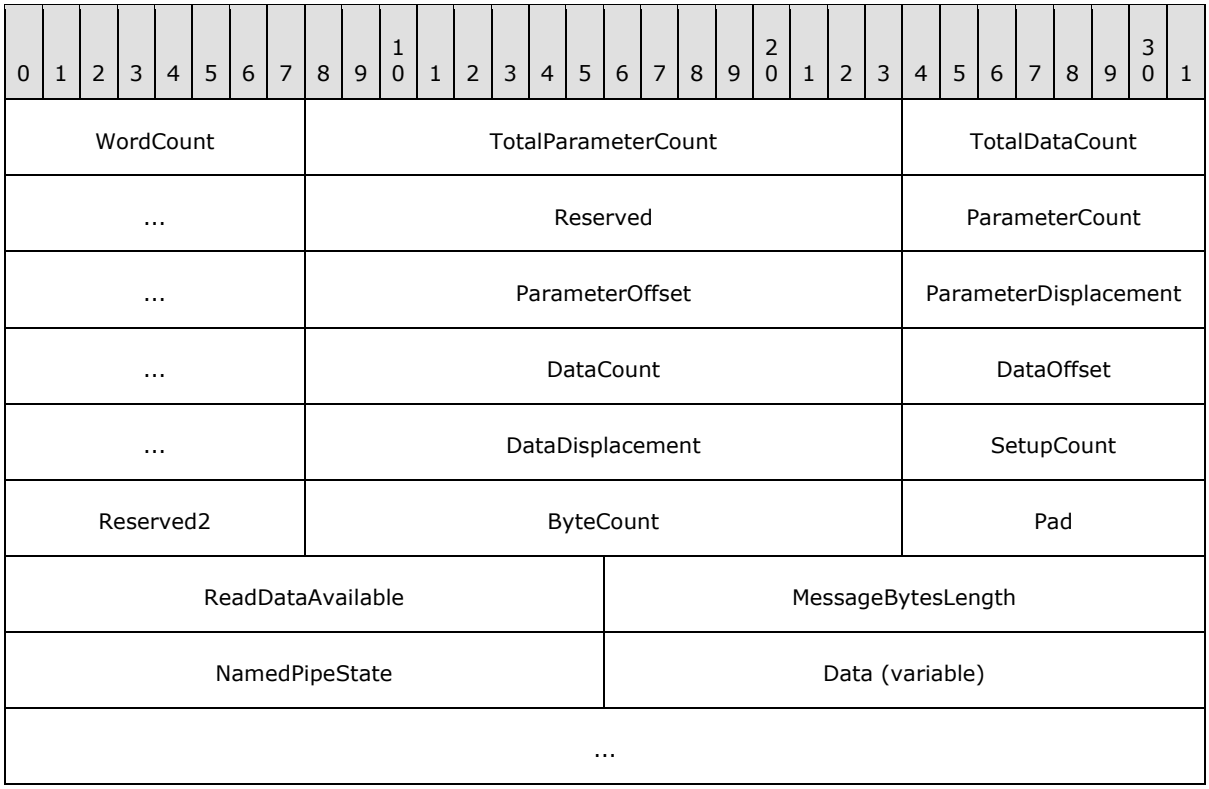
Name (variable): An optional name for this transaction. If SMB_FLAGS2_UNICODE is set in the **Flags2** field of the SMB header of the request, the name string MUST be a null-terminated array of 16-bit Unicode characters. Otherwise, the name string MUST be a null-terminated array of OEM characters. A client MAY set this to the named pipe name. The server MUST ignore this field in the request. If the name string consists of Unicode characters, this field MUST be aligned to start on a 2-byte boundary from the start of the SMB header.<82>

Pad1 (variable): An optional padding to set the Parameter buffer that MUST start on a 4-byte boundary from the start of the SMB header.

2.2.12.10 TRANS_PEEK_NMPIPE Response

A server MUST send a TRANS_PEEK_NMPIPE response in reply to a client TRANS_PEEK_NMPIPE subcommand request when the request is successful. The server MUST set an error code in the **Status** field of the SMB header of the response to indicate whether the query operation on the named pipe was successful or failed.

The format follows the convention for SMB_COM_TRANSACTION, as specified in [CIFS] section 3.13.1.



WordCount (1 byte): The count of 16-bit words in the response structure. For this response, this MUST be set to 0x0A (10), which is 10 plus the **SetupCount** of 0.

TotalParameterCount (2 bytes): The number of bytes in the Parameter buffer. For this response, it MUST be set to 6.

TotalDataCount (2 bytes): The number of bytes in the Data buffer. For this response, it MUST be set to the number of bytes read from the pipe in peek fashion.

Reserved (2 bytes): An unused value that SHOULD be set to 0 when sending this request. The server MUST ignore this field when receiving this message.

ParameterCount (2 bytes): The number of bytes in the Parameter buffer contained in this packet. For this response, it MUST be set to 6.

ParameterOffset (2 bytes): An offset in bytes from the start of the SMB header to the start of the part of the Parameter buffer contained in this response. The Parameter buffer MUST start after the **Reserved2** field. The Parameter buffer MUST be aligned to start on a 4-byte boundary from the start of the SMB header.

ParameterDisplacement (2 bytes): An offset in bytes into the final Parameter buffer assembled from all responses. For a single buffer transaction (whose Parameter buffer fits in a single response), this value MUST be set to 0.

DataCount (2 bytes): The number of bytes in the Data buffer contained in this packet. For this response, it MUST be set to less than or equal to **TotalDataCount**.

DataOffset (2 bytes): An offset in bytes from the start of the SMB header to the start of the Data buffer field in this packet. The Data buffer MUST be greater than or equal to **ParameterOffset** plus **ParameterCount**. The Data buffer MUST be aligned to start on a 4-byte boundary from the start of the SMB header. If the **DataCount** field is 0, the server MAY set the **DataOffset** field to 0, or the server MAY set this field to a location in the packet after where the Parameter bytes would be placed, if present.<83>

DataDisplacement (2 bytes): An offset in bytes into the final Data buffer assembled from all responses. For a single buffer transaction (whose Data buffer fits in a single response), this value MUST be set to 0.

SetupCount (1 byte): The number of setup words. For this response, it MUST be set to 0.

Reserved2 (1 byte): An unused value that SHOULD be set to 0 when sending this response. The client MUST ignore this field when receiving this message.

ByteCount (2 bytes): This value MUST be set to the number of bytes that follows the **ByteCount** field in this packet.

Pad (1 byte): This 1-byte value MUST be set to 0 to pad the start of the Parameter buffer to start on a 4-byte boundary from the start of the SMB header.

ReadDataAvailable (2 bytes): The first USHORT value (as specified in [MS-DTYP] section **2.2.57**) in the Parameter buffer contains the total number of bytes available to be read from the pipe.

MessageBytesLength (2 bytes): If the named pipe is a message mode pipe, this MUST be set to the number of bytes remaining in the message that was peeked (the number of bytes in the message minus the number of bytes read). If the entire message was read, this value is 0. If the named pipe is a byte mode pipe, this value MUST be set to 0.

NamedPipeState (2 bytes): The status of the named pipe.

Value	Meaning
0x0001	Named pipe was disconnected by server.

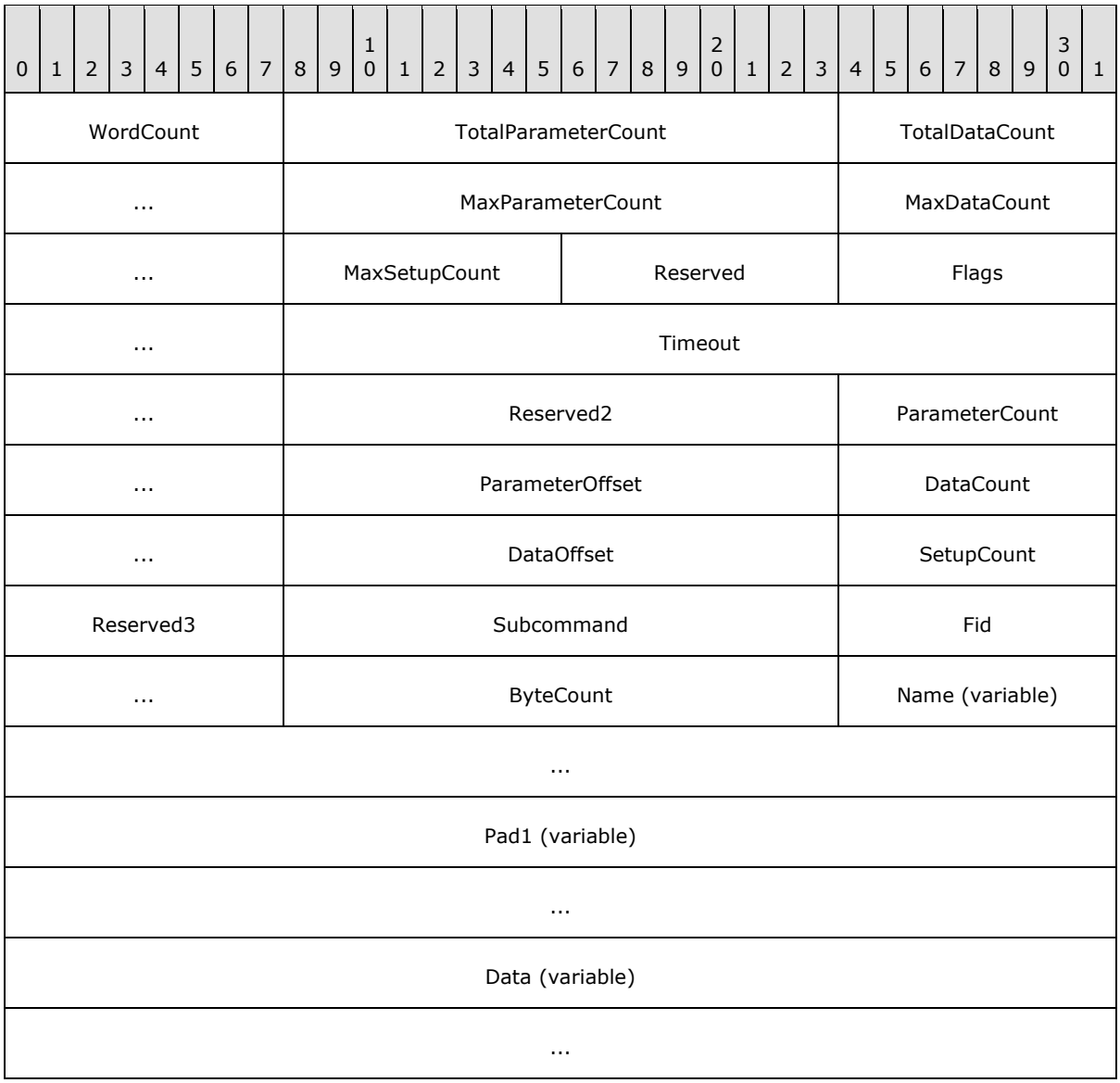
Value	Meaning
0x0002	Named pipe is listening.
0x0003	Named pipe connection to the server is okay.
0x0004	Server end of named pipe is closed.

Data (variable): The data buffer contains the data read from the named pipe.

2.2.12.11 TRANS_TRANSACT_NMPIPE Request

The TRANS_TRANSACT_NMPIPE subcommand of the SMB_COM_TRANSACTION is used to execute a transacted exchange against a named pipe.

The format follows the convention for SMB_COM_TRANSACTION, as specified in [CIFS] section 3.13.1.



WordCount (1 byte): The count of 16-bit words in the request structure. For this request, it MUST be 0x10 (16), which is 14 plus the **SetupCount** of 2.

TotalParameterCount (2 bytes): The number of bytes in the Parameter buffer. For this request, it MUST be set to 0 since there is no parameter buffer associated with this request.

TotalDataCount (2 bytes): The number of bytes to write to the named pipe as part of the transaction.

MaxParameterCount (2 bytes): The maximum number of bytes the client will accept in the Parameter buffer of the response. For this request, it MUST be set to 0 since there is no parameter buffer associated with the response.

MaxDataCount (2 bytes): This value MUST be the number of bytes to read from the named pipe as part of the transaction.

MaxSetupCount (1 byte): The maximum number of bytes the client will accept in the Setup buffer of the response. For this request, it MUST be set to 0.

Reserved (1 byte): An unused value that SHOULD be set to 0 when sending this message. The server MUST ignore this field when receiving this message.

Flags (2 bytes): A set of options that provide additional information to the server on this request. The client MAY set either of the following bits. Unused bit fields SHOULD be set to 0 by the client when sending a request, and MUST be ignored when received by the server.

Value	Meaning
DISCONNECT_TID 0x0001	The server MUST disconnect the tree connect associated with the Tid received in the SMB header of this request after the request completes. The client SHOULD NOT send an SMB_COM_TREE_DISCONNECT for this tree connect.
NO_RESPONSE 0x0002	The server MUST process this client request as a one-way transaction and MUST NOT send a response back to the client.<84>

Timeout (4 bytes): The maximum amount of time in milliseconds to wait for the operation to complete. The client MAY set this to 0 to indicate that no time-out is given. If the operation does not complete within the specified time, the server MAY abort the request and send a failure response.<85>

Reserved2 (2 bytes): An unused value that SHOULD be set to 0 when sending this message. The server MUST ignore this field when receiving this message.

ParameterCount (2 bytes): The count of bytes in the Parameter buffer of this packet. For this request, it MUST be set to 0 since there is no parameter buffer associated with this request.

ParameterOffset (2 bytes): An offset in bytes from the start of the SMB header for this request to the start of the Parameter buffer. The Parameter buffer MUST start after the **Name** field. The Parameter buffer MUST be aligned to start on a 4-byte boundary from the start of the SMB header.

DataCount (2 bytes): The count of bytes in the Data buffer field. For this request, it MUST be set to the number of data bytes in this request to be written to the pipe. For a single-message transaction, this MUST be equal to the **TotalDataCount**.

DataOffset (2 bytes): An offset in bytes to the start of Data buffer field from the start of the SMB header for this request. This value MUST be greater than or equal to **ParameterOffset** plus **ParameterCount**. The Data buffer MUST be aligned to start on a 4-byte boundary from the start of the SMB header. If the **DataCount** field is 0, the client MAY set the **DataOffset** field to 0, or the client MAY set this field to a location in the packet after where the Parameter bytes would be placed, if present.<86>

SetupCount (1 byte): The number of setup words. For this request, it MUST be set to 2.

Reserved3 (1 byte): An unused value that SHOULD be set to 0 when sending this message. The server MUST ignore this field when receiving this message.

Subcommand (2 bytes): The first setup word for this request. This field MUST be set to TRANS_TRANSACT_NMPIPE, the subcommand to request. For a list of named pipe subcommands and values, see section 2.2.12.

Fid (2 bytes): The second setup word for this request. This field is the SMB file identifier for the named pipe that is being transacted. This field MUST be set to a valid Fid from a server

response for a previous SMB command to open or create a named pipe. These commands include SMB_COM_OPEN, SMB_COM_CREATE, SMB_COM_CREATE_TEMPORARY, SMB_COM_CREATE_NEW, SMB_COM_OPEN_ANDX, SMB_COM_NT_CREATE_ANDX, and SMB_COM_NT_TRANSACT with subcommand NT_TRANSACT_CREATE.

ByteCount (2 bytes): This value MUST be the number of bytes that follows the **ByteCount** field in this packet.

Name (variable): An optional name for this transaction. If SMB_FLAGS2_UNICODE is set in the **Flags2** field of the SMB header of the request, the name string MUST be a null-terminated array of 16-bit Unicode characters. Otherwise, the name string MUST be a null-terminated array of OEM characters. A client MAY set this to the named pipe name. The server MUST ignore this field in the request. If the name string consists of Unicode characters, this field MUST be aligned to start on a 2-byte boundary from the start of the SMB header.<87>

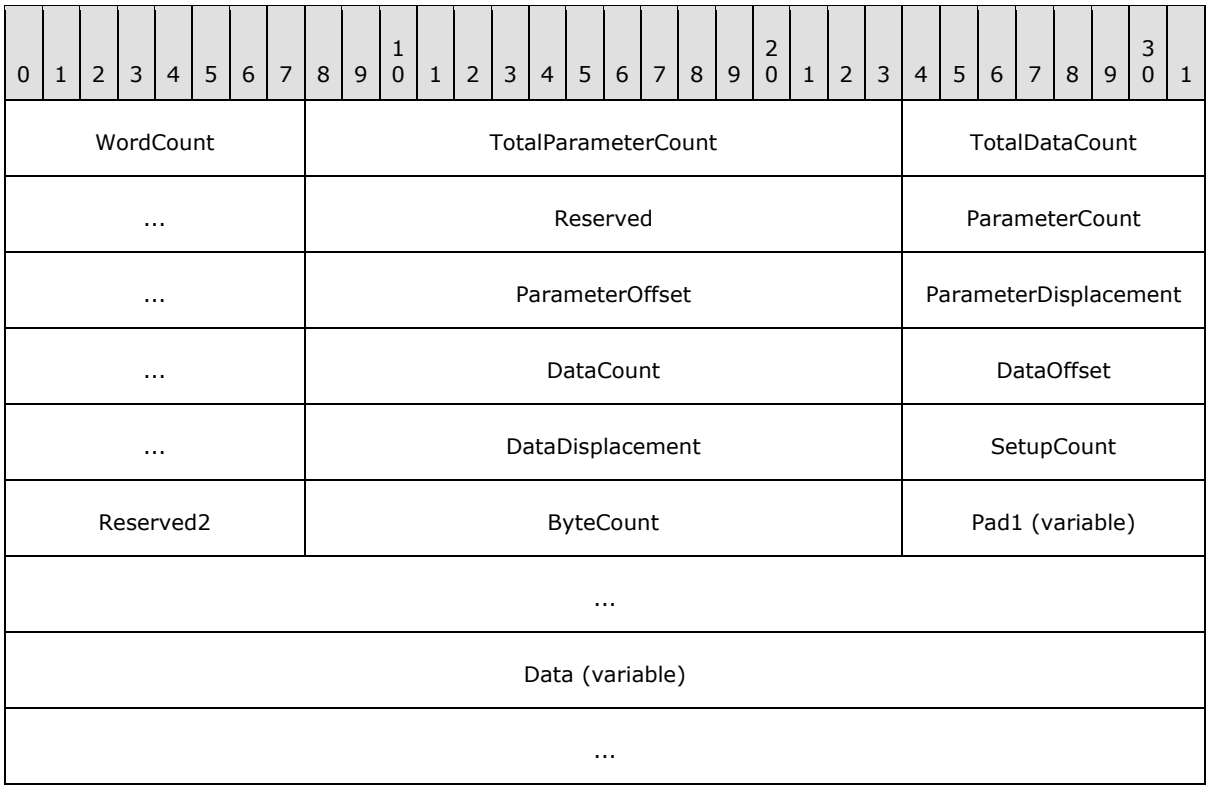
Pad1 (variable): Variable length padding SHOULD be initialized with zeros to set the Parameter buffer to start on a 4-byte boundary from the start of the SMB header. The server MUST ignore the padding.

Data (variable): Data buffer containing the bytes to be written to the pipe as part of the transacted operation.

2.2.12.12 TRANS_TRANSACT_NMPIPE Response

The server MUST send a TRANS_TRANSACT_NMPIPE in reply to a client TRANS_TRANSACT_NMPIPE subcommand request when the request is successful. The server MUST set an error code in the **Status** field of the SMB header of the response to indicate whether the transact operation on the named pipe was successful or failed.

The format follows the convention for SMB_COM_TRANSACTION, as specified in [CIFS] section 3.13.1.



WordCount (1 byte): The count of 16-bit words in the response structure. For this response, this MUST be set to 0x0A (10), which is 10 plus the **SetupCount** of 0.

TotalParameterCount (2 bytes): The number of bytes in the Parameter buffer. For this response, it MUST be set to 0 since there is no parameter buffer associated with this response.

TotalDataCount (2 bytes): The number of bytes in the Data buffer. For this response, it MUST be set to the number of bytes read from the pipe.

Reserved (2 bytes): An unused value that SHOULD be set to 0 when sending this response. The client MUST ignore this field when receiving this message.

ParameterCount (2 bytes): The number of bytes in the Parameter buffer contained in this packet. For this response, it MUST be set to 0 since there is no parameter buffer associated with this response.

ParameterOffset (2 bytes): An offset in bytes from the start of the SMB header to the start of the part of the Parameter buffer contained in this response. The Parameter buffer MUST start after the **Reserved2** field. The Parameter buffer MUST be aligned to start on a 4-byte boundary from the start of the SMB header. If the **ParameterCount** field is 0, the client MAY set the **ParameterOffset** field to 0, or the client MAY set this field to a location in the packet after the **Name** field where Parameter bytes would have been placed, if present.<88>

ParameterDisplacement (2 bytes): An offset in bytes into the final Parameters buffer assembled from all responses. For a single buffer transaction (whose Parameters buffer fits in a single response), this value MUST be set to 0.

DataCount (2 bytes): The number of bytes in the Data buffer contained in this packet. For this response, it MUST be less than or equal to **TotalDataCount**.

DataOffset (2 bytes): An offset in bytes from the start of the SMB header to the start of the part of the Data buffer contained in this response. This value MUST be greater than **ParameterOffset** plus **ParameterCount**. The Data buffer MUST be aligned to start on a 4-byte boundary from the start of the SMB header.

DataDisplacement (2 bytes): An offset in bytes into the final Data buffer assembled from all responses. For a single buffer transaction (whose Data buffer fits in a single response), this value MUST be set to 0.

SetupCount (1 byte): The number of setup words. For this response, it MUST be set to 0.

Reserved2 (1 byte): An unused value that SHOULD be set to 0 when sending this response. The client MUST ignore this field when receiving this message.

ByteCount (2 bytes): This value MUST be set to the number of bytes that follows the **ByteCount** field in this packet.

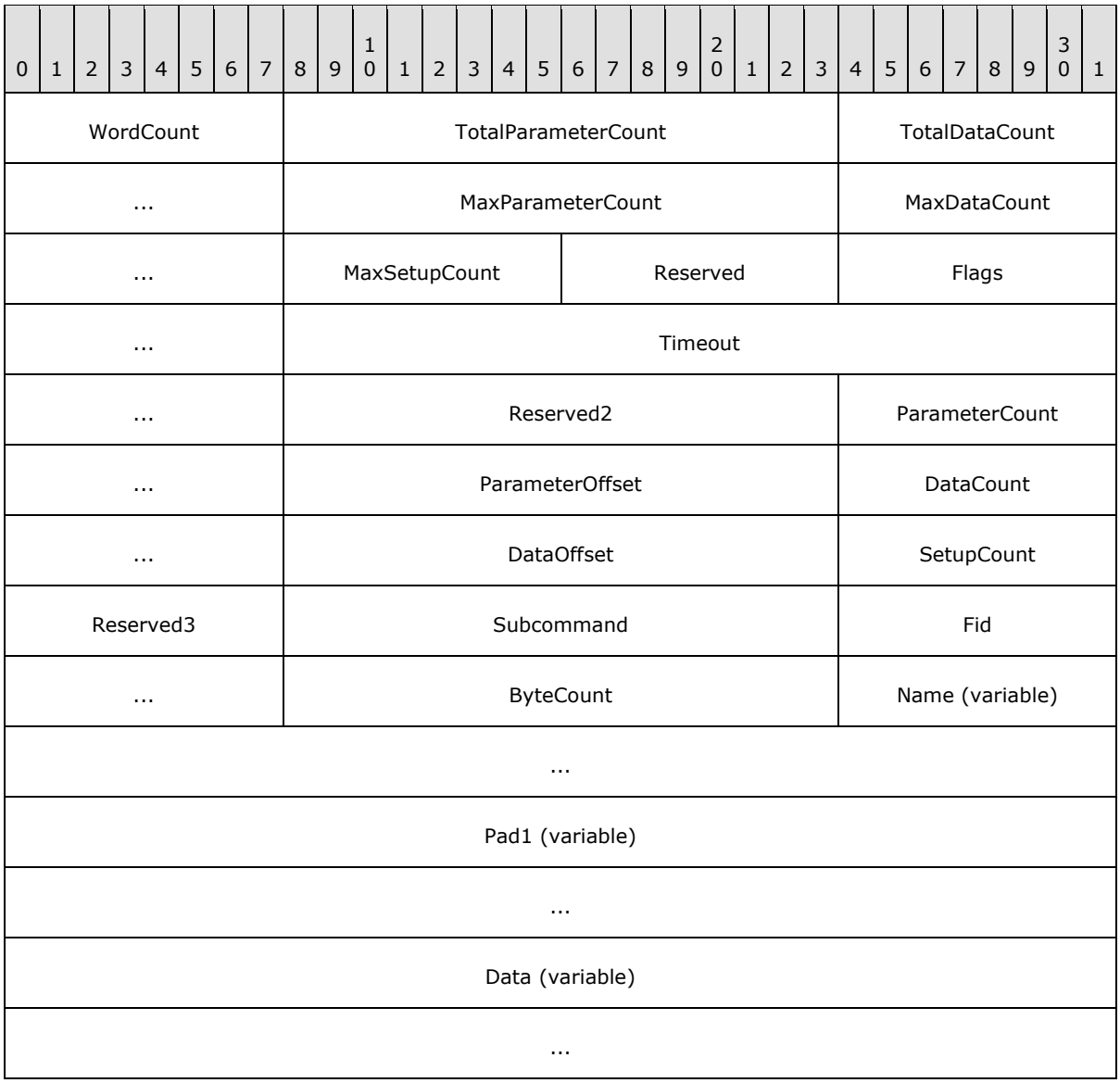
Pad1 (variable): A potential padding to set the start of the Parameter buffer that MUST be set on a 4-byte boundary from the start of the SMB header.

Data (variable): The Data buffer that contains the data read from the named pipe. If the entire response message was not read from the named pipe, the server MUST return STATUS_BUFFER_OVERFLOW in the **Status** field of the SMB header along with the data in the Data buffer field that was read from the named pipe.

2.2.12.13 TRANS_RAW_WRITE_NMPIPE Request

The TRANS_RAW_WRITE_NMPIPE subcommand of the SMB_COM_TRANSACTION allows for a **raw write** of data to a named pipe. Raw writes to named pipes put bytes directly into a pipe, regardless of whether it is a message mode pipe or stream mode pipe. The method of writing data into a named pipe requires that the data MUST contain the message boundaries if the pipe is a message mode pipe. The operation can allow a single write to insert multiple messages.<89>

The format follows the convention for SMB_COM_TRANSACTION, as specified in [CIFS] section 3.13.1.



WordCount (1 byte): The count of 16-bit words in the request structure. For this request, it MUST be 0x10 (16), which is 14 plus the SetupCount of 2.

TotalParameterCount (2 bytes): The number of bytes in the Parameter buffer. For this request, it MUST be 0.

TotalDataCount (2 bytes): The number of bytes in the Data buffer. For this request, it MUST be the number of bytes that are being written to the pipe in raw format.

MaxParameterCount (2 bytes): The maximum number of bytes the client will accept in the Parameter buffer of the response. For this request, it MUST be set to 0.

MaxDataCount (2 bytes): The maximum number of bytes the client will accept in the Data buffer of the response. For this request, it MUST be set to 0.

MaxSetupCount (1 byte): The maximum number of bytes the client will accept in the Setup buffer of the response. For this request, it MUST be set to 0.

Reserved (1 byte): An unused value that SHOULD be set to 0 when sending this message. The server MUST ignore this field when receiving this message.

Flags (2 bytes): A set of options that provide additional information to the server on this request. The client MAY set either of the following bits. Unused bit fields SHOULD be set to 0 by the client when sending a request, and MUST be ignored when received by the server.

Value	Meaning
DISCONNECT_TID 0x0001	The server MUST disconnect the tree connect associated with the Tid received in the SMB header of this request after the request completes. The client SHOULD NOT send an SMB_COM_TREE_DISCONNECT for this tree connect.
NO_RESPONSE 0x0002	The server MUST process this client request as a one-way transaction and MUST NOT send a response back to the client.<90>

Timeout (4 bytes): The maximum amount of time in milliseconds to wait for the operation to complete. The client MAY set this to 0 to indicate that no time-out is given. If the operation does not complete within the specified time, the server MAY abort the request and send a failure response.<91>

Reserved2 (2 bytes): An unused value that SHOULD be set to 0 when sending this message. The server MUST ignore this field when receiving this message.

ParameterCount (2 bytes): The count of bytes in the Parameter buffer of this packet. For this request, it MUST be set to 0.

ParameterOffset (2 bytes): An offset in bytes from the start of the SMB header for this request to the start of the Parameter buffer. The Parameter buffer MUST start after the **Name** field. The Parameter buffer MUST be aligned to start on a 4-byte boundary from the start of the SMB header. If the **ParameterCount** field is 0, the client MAY set the **ParameterOffset** field to 0, or the client MAY set this field to a location in the packet after the **Name** field where Parameter bytes would have been placed, if present.<92>

DataCount (2 bytes): The number of bytes in the Data buffer field. For this request, it MUST be set to the number of bytes being written to the pipe in raw format contained in this request.

DataOffset (2 bytes): An offset in bytes to the start of Data buffer field from the start of the SMB header for this request. This value MUST be greater than or equal to **ParameterOffset** plus **ParameterCount**. The Data buffer MUST be aligned to start on a 4-byte boundary from the start of the SMB header.

SetupCount (1 byte): The number of setup words. For this request, it MUST be set to 2.

Reserved3 (1 byte): An unused value that SHOULD be set to 0 when sending this message. The server MUST ignore this field when receiving this message.

Subcommand (2 bytes): The first setup word for this request. This field MUST be set to TRANS_RAW_WRITE_NMPIPE, the subcommand to request. For a list of named pipe subcommands and values, see section 2.2.12.

Fid (2 bytes): The second setup word for this request. This field is the SMB file identifier for the named pipe to write data to. This field MUST be set to a valid Fid from a server response

for a previous SMB command to open or create a named pipe. These commands include SMB_COM_OPEN, SMB_COM_CREATE, SMB_COM_CREATE_TEMPORARY, SMB_COM_CREATE_NEW, SMB_COM_OPEN_ANDX, SMB_COM_NT_CREATE_ANDX, and SMB_COM_NT_TRANSACT with subcommand NT_TRANSACT_CREATE.

ByteCount (2 bytes): This value MUST be the number of bytes that follows the **ByteCount** field in this packet.

Name (variable): An optional name for this transaction. If SMB_FLAGS2_UNICODE is set in the **Flags2** field of the SMB header of the request, the name string MUST be a null-terminated array of 16-bit Unicode characters. Otherwise, the name string MUST be a null-terminated array of OEM characters. A client MAY set this to the named pipe name. The server MUST ignore this field in the request. If the name string consists of Unicode characters, this field MUST be aligned to start on a 2-byte boundary from the start of the SMB header.<93>

Pad1 (variable): Variable length padding SHOULD be initialized with zeros to set the data buffer to start on a 4-byte boundary from the start of the SMB header. The server MUST ignore the padding.

Data (variable): Data buffer that contains the bytes to write to the named pipe in raw format.

2.2.12.14 TRANS_RAW_WRITE_NMPIPE Response

This message MUST be sent by a server to respond to a client TRANS_RAW_WRITE_NMPIPE subcommand request when the request is successful. The server MUST set an error code in the **Status** field of the SMB header of the response to indicate whether the write request on the named pipe was successful or failed.

The format follows the convention for SMB_COM_TRANSACTION, as specified in [CIFS] section 3.13.1.

0	1	2	3	4	5	6	7	8	9	10	1	2	3	4	5	6	7	8	9	20	1	2	3	4	5	6	7	8	9	30	1
WordCount								TotalParameterCount																TotalDataCount							
...								Reserved																ParameterCount							
...								ParameterOffset																ParameterDisplacement							
...								DataCount																DataOffset							
...								DataDisplacement																SetupCount							
Reserved 2								ByteCount																Pad							

WordCount (1 byte): The count of 16-bit words in the response structure. For this response, this MUST be set to 0x0A (10), which is 10 plus the **SetupCount** of 0.

TotalParameterCount (2 bytes): The number of bytes in the Parameter buffer. For this response, it MUST be set to 0.

TotalDataCount (2 bytes): The number of bytes in the Data buffer. For this response, it MUST be set to 0.

Reserved (2 bytes): An unused value that SHOULD be set to 0 when sending this response. The client MUST ignore this field when receiving this message.

ParameterCount (2 bytes): The number of bytes in the Parameter buffer contained in this packet. For this response, it MUST be set to 0.

ParameterOffset (2 bytes): An offset in bytes from the start of the SMB header to the start of the part of the Parameter buffer contained in this response. The Parameter buffer MUST start after the **Reserved2** field. The Parameter buffer MUST be aligned to start on a 4-byte boundary from the start of the SMB header. If the **ParameterCount** field is 0, the server MAY set the **ParameterOffset** field to 0 or the server MAY set this field to a location in the packet after where Parameter bytes would have been placed if present.<94>

ParameterDisplacement (2 bytes): An offset in bytes into the final Parameters buffer assembled from all responses. For a single buffer transaction (whose Parameters buffer fits in a single response), this value MUST be set to 0.

DataCount (2 bytes): The number of bytes in the Data buffer contained in this packet. For this response, it MUST be set to 0.

DataOffset (2 bytes): An offset in bytes from the start of the SMB header to the start of the part of the Data buffer contained in this response. This value MUST be greater than **ParameterOffset** plus **ParameterCount**. The Data buffer MUST be aligned to start on a 4-byte boundary from the start of the SMB header. If the **DataCount** field is 0, the server MAY set the **DataOffset** field to 0 or the server MAY set this field to a location in the packet after where the Parameter bytes would be placed if present.<95>

DataDisplacement (2 bytes): An offset in bytes into the final Data buffer assembled from all responses. For a single buffer transaction (whose Data buffer fits in a single response), this value MUST be set to 0.

SetupCount (1 byte): The number of setup words. For this request, it MUST be set to 0.

Reserved 2 (1 byte): An unused value that SHOULD be set to 0 when sending this response. The client MUST ignore this field when receiving this message.

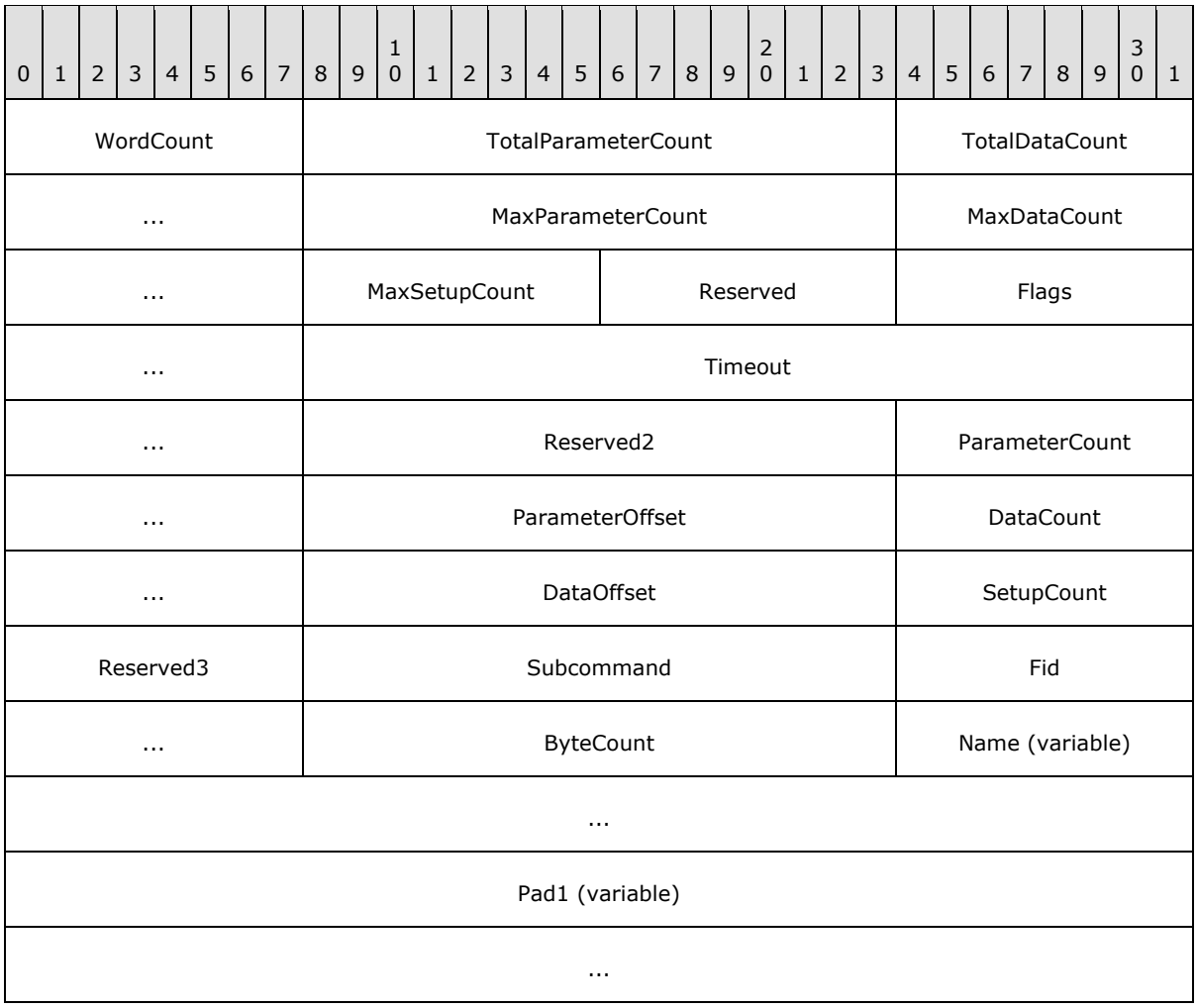
ByteCount (2 bytes): This value MUST be set to the number of bytes that follows the **ByteCount** field in this packet.

Pad (1 byte): One byte of padding to set the start of the Parameter buffer on a 4-byte boundary from the start of the SMB header. The server SHOULD set this padding to 0 and the client MUST ignore it.

2.2.12.15 TRANS_READ_NMPIPE Request

The TRANS_READ_NMPIPE subcommand of the SMB_COM_TRANSACTION allows a client to read data from a named pipe.

The format follows the convention for SMB_COM_TRANSACTION, as specified in [CIFS] section 3.13.1.



WordCount (1 byte): The count of 16-bit words in the request structure. For this request, it MUST be 0x10 (16), which is 14 plus the **SetupCount** of 2.

TotalParameterCount (2 bytes): The number of bytes in the Parameter buffer. For this request, it MUST be 2. The content of the Parameter buffer is described below.

TotalDataCount (2 bytes): The number of bytes in the Data buffer. For this request, it MUST be set to 0.

MaxParameterCount (2 bytes): The maximum number of bytes the client will accept in the Parameter buffer of the response. For this request, it MUST be set to 0.

MaxDataCount (2 bytes): The maximum number of bytes the client will accept in the Data buffer of the response. For this request, it MUST be set to the number of bytes the client is attempting to read from the named pipe.

MaxSetupCount (1 byte): The maximum number of bytes the client will accept in the Setup buffer of the response. For this request, it MUST be set to 0.

Reserved (1 byte): An unused value that SHOULD be set to 0 when sending this message. The server MUST ignore this field when receiving this message.

Flags (2 bytes): A set of options that provide additional information to the server on this request. The client MAY set either of the following bits. Unused bit fields SHOULD be set to 0 by the client when sending a request, and MUST be ignored when received by the server.

Value	Meaning
DISCONNECT_TID 0x0001	The server MUST disconnect the tree connect associated with the Tid received in the SMB header of this request after the request completes. The client SHOULD NOT send an SMB_COM_TREE_DISCONNECT for this tree connect.
NO_RESPONSE 0x0002	The server MUST process this client request as a one-way transaction and MUST NOT send a response back to the client.<96>

Timeout (4 bytes): The maximum amount of time in milliseconds to wait for the operation to complete. The client MAY set this to 0 to indicate that no time-out is given. If the operation does not complete within the specified time, the server MAY abort the request and send a failure response.<97>

Reserved2 (2 bytes): An unused value that SHOULD be set to 0 when sending this message. The SMB server MUST ignore this field when receiving this message.

ParameterCount (2 bytes): The count of bytes in the Parameter buffer of this packet. For this request, it MUST be set to 0.

ParameterOffset (2 bytes): An offset in bytes from the start of the SMB header for this request to the start of the Parameter buffer. The Parameter buffer MUST start after the **Name** field. The Parameter buffer MUST be aligned to start on a 4-byte boundary from the start of the SMB header. If the **ParameterCount** field is 0, the client MAY set the **ParameterOffset** field to 0, or the client MAY set this field to a location in the packet after the **Name** field where Parameter bytes would have been placed, if present.<98>

DataCount (2 bytes): The count of bytes in the Data buffer field. For this request, it MUST be set to 0.

DataOffset (2 bytes): An offset in bytes to the start of Data buffer field from the start of the SMB header for this request. This value MUST be greater than or equal to **ParameterOffset** plus **ParameterCount**. The Data buffer MUST be aligned to start on a 4-byte boundary from the start of the SMB header. If the **DataCount** field is 0, the client MAY set the **DataOffset** field to 0, or the client MAY set this field to a location in the packet after where the Parameter bytes would be placed, if present.<99>

SetupCount (1 byte): The number of setup words. For this request, it MUST be set to 2.

Reserved3 (1 byte): An unused value that SHOULD be set to 0 when sending this message. The server MUST ignore this field when receiving this message.

Subcommand (2 bytes): The first setup word for this request. This field MUST be set to TRANS_READ_NMPIPE, the subcommand to request. For a list of named pipe subcommands and values, see section 2.2.12.

Fid (2 bytes): The second setup word for this request. This field is the SMB file identifier for the named pipe to read data from. This field MUST be set to a valid Fid from a server response for a previous SMB command to open or create a named pipe. These commands include SMB_COM_OPEN, SMB_COM_CREATE, SMB_COM_CREATE_TEMPORARY,

SMB_COM_CREATE_NEW, SMB_COM_OPEN_ANDX, SMB_COM_NT_CREATE_ANDX, and SMB_COM_NT_TRANSACT with subcommand NT_TRANSACT_CREATE.

ByteCount (2 bytes): This value MUST be the number of bytes that follows the **ByteCount** field in this packet.

Name (variable): An optional name for this transaction. If SMB_FLAGS2_UNICODE is set in the **Flags2** field of the SMB header of the request, the name string MUST be a null-terminated array of 16-bit Unicode characters. Otherwise, the name string MUST be a null-terminated array of OEM characters. A client MAY set this to the named pipe name. The server MUST ignore this field in the request. If the name string consists of Unicode characters, this field MUST be aligned to start on a 2-byte boundary from the start of the SMB header.<100>

Pad1 (variable): The variable-length padding SHOULD be initialized with zeros to set the parameter buffer to start on a 4-byte boundary from the start of the SMB header. The server MUST ignore the padding.

2.2.12.16 TRANS_READ_NMPIPE Response

This message MUST be sent by a server to respond to a client TRANS_READ_NMPIPE request when the request is successful. The server MUST set an error code in the **Status** field of the SMB header of the response to indicate whether the read operation on the named pipe was successful or failed.

The format follows the convention for SMB_COM_TRANSACTION, as specified in [CIFS] section 3.13.1.

0	1	2	3	4	5	6	7	8	9	10	1	2	3	4	5	6	7	8	9	20	1	2	3	4	5	6	7	8	9	30	1
WordCount								TotalParameterCount																TotalDataCount							
...								Reserved																ParameterCount							
...								ParameterOffset																ParameterDisplacement							
...								DataCount																DataOffset							
...								DataDisplacement																SetupCount							
Reserved2								ByteCount																Pad1 (optional)							
Data (variable)																															
...																															

WordCount (1 byte): The count of 16-bit words in the response structure. For this response, this MUST be set to 0x0A (10), which is 10 plus the **SetupCount** value of 0.

TotalParameterCount (2 bytes): The number of bytes in the Parameter buffer. For this response, it MUST be set to 0.

TotalDataCount (2 bytes): The number of bytes in the Data buffer. For this response, it MUST be set to the number of bytes read from the named pipe.

Reserved (2 bytes): An unused value that SHOULD be set to 0 when sending this response. The client MUST ignore this field when receiving this message.

ParameterCount (2 bytes): The number of bytes in the Parameter buffer contained in this packet. For this response, it MUST be set to 0.

ParameterOffset (2 bytes): An offset in bytes from the start of the SMB header to the start of the part of the Parameter buffer contained in this response. The Parameter buffer MUST start after the **Reserved2** field. The Parameter buffer MUST be aligned to start on a 4-byte boundary from the start of the SMB header. If the **ParameterCount** field is 0, the server MAY set the **ParameterOffset** field to 0 or the server MAY set this field to a location in the packet after where Parameter bytes would have been placed if present.<101>

ParameterDisplacement (2 bytes): An offset in bytes into the final Parameters buffer assembled from all responses. For a single buffer transaction (whose Parameters buffer fits in a single response), this value MUST be set to 0.

DataCount (2 bytes): The number of bytes in the Data buffer contained in this packet. For this response, it MUST be set to less than or equal to **TotalDataCount**.

DataOffset (2 bytes): An offset in bytes from the start of the SMB header to the start of the part of the Data buffer contained in this response. This value MUST be greater than **ParameterOffset** plus **ParameterCount**. The Data buffer MUST be aligned to start on a 4-byte boundary from the start of the SMB header. If the **DataCount** field is 0, the server MAY set the **DataOffset** field to 0, or the server MAY set this field to a location in the packet after where the Parameter bytes would be placed, if present.<102>

DataDisplacement (2 bytes): An offset in bytes into the final Data buffer assembled from all responses. For a single buffer transaction (whose Data buffer fits in a single response), this value MUST be set to 0.

SetupCount (1 byte): The number of setup words. For this request, it MUST be set to 0.

Reserved2 (1 byte): An unused value that SHOULD be set to 0 when sending this response. The client MUST ignore this field when receiving this message.

ByteCount (2 bytes): This value MUST be set to the number of bytes that follows the **ByteCount** field in this packet.

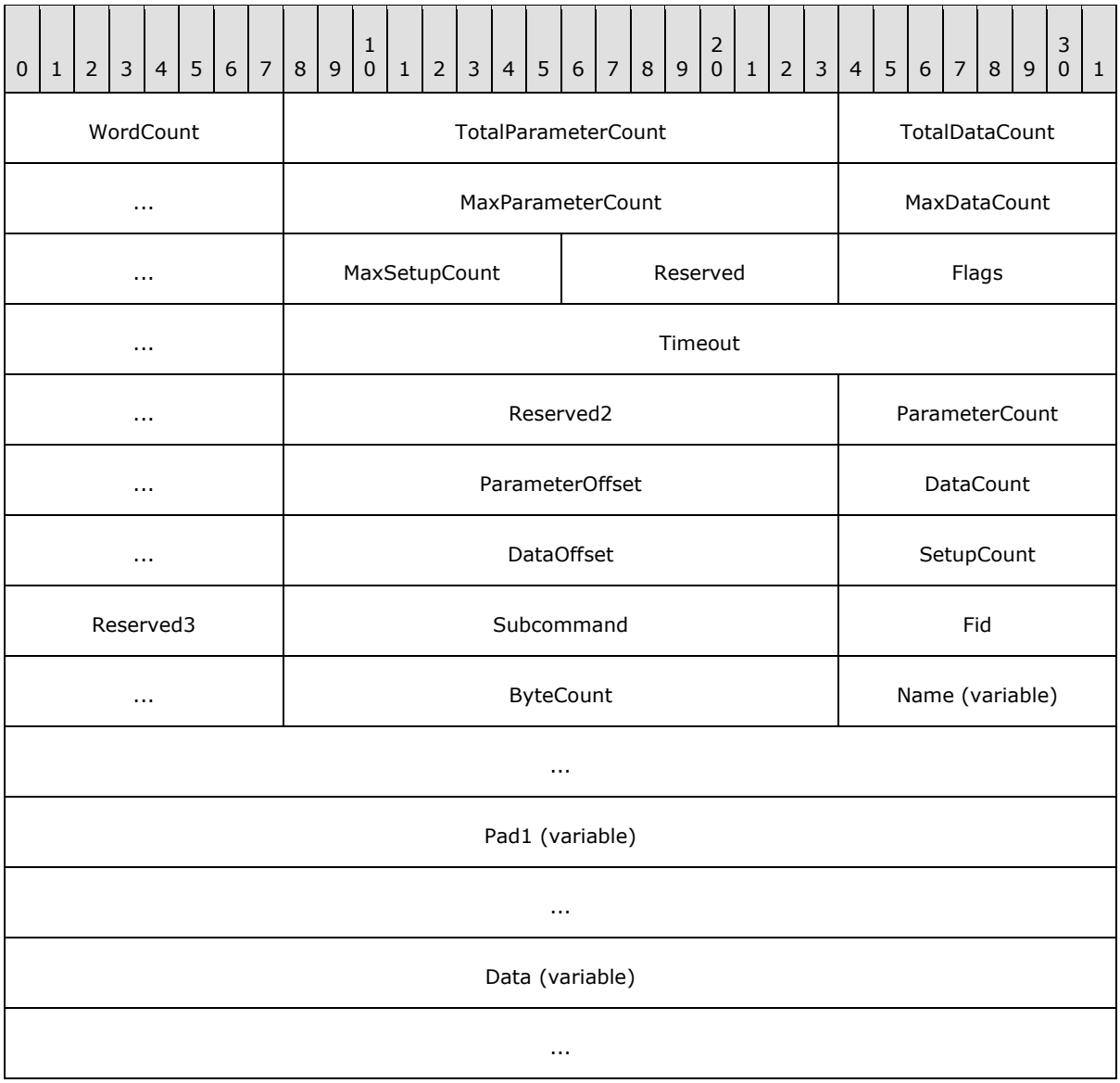
Pad1 (1 byte): A potential padding to set the start of the Parameter buffer that MUST be on a 4-byte boundary from the start of the SMB header.

Data (variable): Data buffer that contains the bytes read from the named pipe. If the named pipe is a message mode pipe, and the entire message was not read, the **Status** field in the SMB header MUST be set to STATUS_BUFFER_OVERFLOW.

2.2.12.17 TRANS_WRITE_NMPIPE Request

The TRANS_WRITE_NMPIPE subcommand of the SMB_COM_TRANSACTION allows a client to write data to a named pipe.

The format follows the convention for SMB_COM_TRANSACTION, as specified in [CIFS] section 3.13.1.



WordCount (1 byte): The count of 16-bit words in the request structure. For this request, it MUST be 0x10 (16), which is 14 plus the SetupCount of 2.

TotalParameterCount (2 bytes): The number of bytes in the Parameter buffer. For this request, it MUST be 2. The content of the Parameter buffer is described below.

TotalDataCount (2 bytes): The number of bytes in the Data buffer. For this request, it MUST be the number of bytes to write to the named pipe.

MaxParameterCount (2 bytes): The maximum number of bytes the client will accept in the Parameter buffer of the response. For this request, it MUST be set to 0.

MaxDataCount (2 bytes): The maximum number of bytes the client will accept in the Data buffer of the response. For this request, it MUST be set to 0.

MaxSetupCount (1 byte): The maximum number of bytes the client will accept in the Setup buffer of the response. For this request, it MUST be set to 0.

Reserved (1 byte): An unused value that SHOULD be set to 0 when sending this message. The server SHOULD ignore this field when receiving this message.

Flags (2 bytes): A set of options that provide additional information to the server on this request. The client MAY set either of the following bits. Unused bit fields SHOULD be set to 0 by the client when sending a request, and MUST be ignored when received by the server.

Value	Meaning
DISCONNECT_TID 0x0001	The server MUST disconnect the tree connect associated with the Tid received in the SMB header of this request after the request completes. The client SHOULD NOT send an SMB_COM_TREE_DISCONNECT for this tree connect.
NO_RESPONSE 0x0002	The server MUST process this client request as a one-way transaction and MUST NOT send a response back to the client.<103>

Timeout (4 bytes): MUST be the maximum amount of time in milliseconds to wait for the operation to complete. The client MAY set this to 0 to indicate that no time-out is given. If the operation does not complete within the specified time, the server MAY abort the request and send a failure response.<104>

Reserved2 (2 bytes): An unused value that SHOULD be set to 0 when sending this message. The server MUST ignore this field when receiving this message.

ParameterCount (2 bytes): The count of bytes in the Parameter buffer of this packet. For this request, it MUST be 2.

ParameterOffset (2 bytes): An offset in bytes from the start of the SMB header for this request to the start of the Parameter buffer. The Parameter buffer MUST start after the **Name** field. The Parameter buffer MUST be aligned to start on a 4-byte boundary from the start of the SMB header.

DataCount (2 bytes): The count of bytes in the Data buffer field. For this request, it MUST be set to the number of bytes in this packet to write to the pipe.

DataOffset (2 bytes): An offset in bytes to the start of Data buffer field from the start of the SMB header for this request. This value MUST be greater than or equal to **ParameterOffset** plus **ParameterCount**. The Data buffer MUST be aligned to start on a 4-byte boundary from the start of the SMB header.

SetupCount (1 byte): The number of setup words. For this request, it MUST be set to 2.

Reserved3 (1 byte): An unused value that SHOULD be set to 0 when sending this message. The server MUST ignore this field when receiving this message.

Subcommand (2 bytes): The first setup word for this request. This field MUST be set to TRANS_WRITE_NMPIPE, the subcommand to request. For a list of named pipe subcommands and values, see section 2.2.12.

Fid (2 bytes): The second setup word for this request. This field is the SMB file identifier for the named pipe to write data to. This field MUST be set to a valid Fid from a server response for a previous SMB command to open or create a named pipe. These commands include SMB_COM_OPEN, SMB_COM_CREATE, SMB_COM_CREATE_TEMPORARY,

SMB_COM_CREATE_NEW, SMB_COM_OPEN_ANDX, SMB_COM_NT_CREATE_ANDX, and SMB_COM_NT_TRANSACT with subcommand NT_TRANSACT_CREATE.

ByteCount (2 bytes): This value MUST be the number of bytes that follows the **ByteCount** field in this packet.

Name (variable): An optional name for this transaction. If SMB_FLAGS2_UNICODE is set in the <mark type="field">Flags2</mark> field of the SMB header of the request, the name string MUST be a null-terminated array of 16-bit Unicode characters. Otherwise, the name string MUST be a null-terminated array of OEM characters. A client MAY set this to the named pipe name. The server MUST ignore this field in the request. If the name string consists of Unicode characters, this field MUST be aligned to start on a 2-byte boundary from the start of the SMB header.<105>

Pad1 (variable): An optional padding to set the Parameter buffer (the **PipeState** field) to start on a 4-byte boundary from the start of the SMB header.

Data (variable): Data buffer that contains the data to write to the named pipe.

2.2.12.18 TRANS_WRITE_NMPIPE Response

This message MUST be sent by a server to respond to a client TRANS_WRITE_NMPIPE subcommand request when the request is successful. The server MUST set an error code in the **Status** field of SMB header of the response to indicate whether the transact operation on the named pipe was successful or failed.

The format follows the convention for SMB_COM_TRANSACTION, as specified in [CIFS] section 3.13.1.

0	1	2	3	4	5	6	7	8	9	10	1	2	3	4	5	6	7	8	9	20	1	2	3	4	5	6	7	8	9	30	1
WordCount								TotalParameterCount																TotalDataCount							
...								Reserved																ParameterCount							
...								ParameterOffset																ParameterDisplacement							
...								DataCount																DataOffset							
...								DataDisplacement																SetupCount							
Reserved 2								ByteCount																Pad1 (optional)							

WordCount (1 byte): The count of 16-bit words in the response structure. For this response, this MUST be set to 0x0A (10), which is 10 plus the **SetupCount** of 0.

TotalParameterCount (2 bytes): The number of bytes in the Parameter buffer. For this response, it MUST be set to 0.

TotalDataCount (2 bytes): The number of bytes in the Data buffer. For this response, it MUST be set to 0.

Reserved (2 bytes): An unused value that SHOULD be set to 0 when sending this response. The client MUST ignore this field when receiving this message.

ParameterCount (2 bytes): The number of bytes in the Parameter buffer contained in this packet. For this response, it MUST be set to 0.

ParameterOffset (2 bytes): An offset in bytes from the start of the SMB header to the start of the part of the Parameter buffer contained in this response. The Parameter buffer MUST start after the **Reserved2** field. The Parameter buffer MUST be aligned to start on a 4-byte boundary from the start of the SMB header. If the **ParameterCount** field is 0, the server MAY set the **ParameterOffset** field to 0, or the server MAY set this field to a location in the packet where Parameter bytes would have been placed, if present.<106>

ParameterDisplacement (2 bytes): An offset in bytes into the final Parameters buffer assembled from all responses. For a single buffer transaction (whose Parameters buffer fits in a single response), this value MUST be set to 0.

DataCount (2 bytes): The number of bytes in the Data buffer contained in this packet. For this response, it MUST be set to 0.

DataOffset (2 bytes): An offset in bytes from the start of the SMB header to the start of the part of the Data buffer contained in this response. This value MUST be greater than **ParameterOffset** plus **ParameterCount**. The Data buffer MUST be aligned to start on a 4-byte boundary from the start of the SMB header. If the **DataCount** field is 0, the server MAY set the **DataOffset** field to 0 ,or the server MAY set this field to a location in the packet after where the Parameter bytes would be placed, if present.<107>

DataDisplacement (2 bytes): An offset in bytes into the final Data buffer assembled from all responses. For a single buffer transaction (whose Data buffer fits in a single response), this value MUST be set to 0.

SetupCount (1 byte): The number of setup words. For this request, it MUST be set to 0.

Reserved 2 (1 byte): An unused value that SHOULD be set to 0 when sending this response. The client MUST ignore this field when receiving this message.

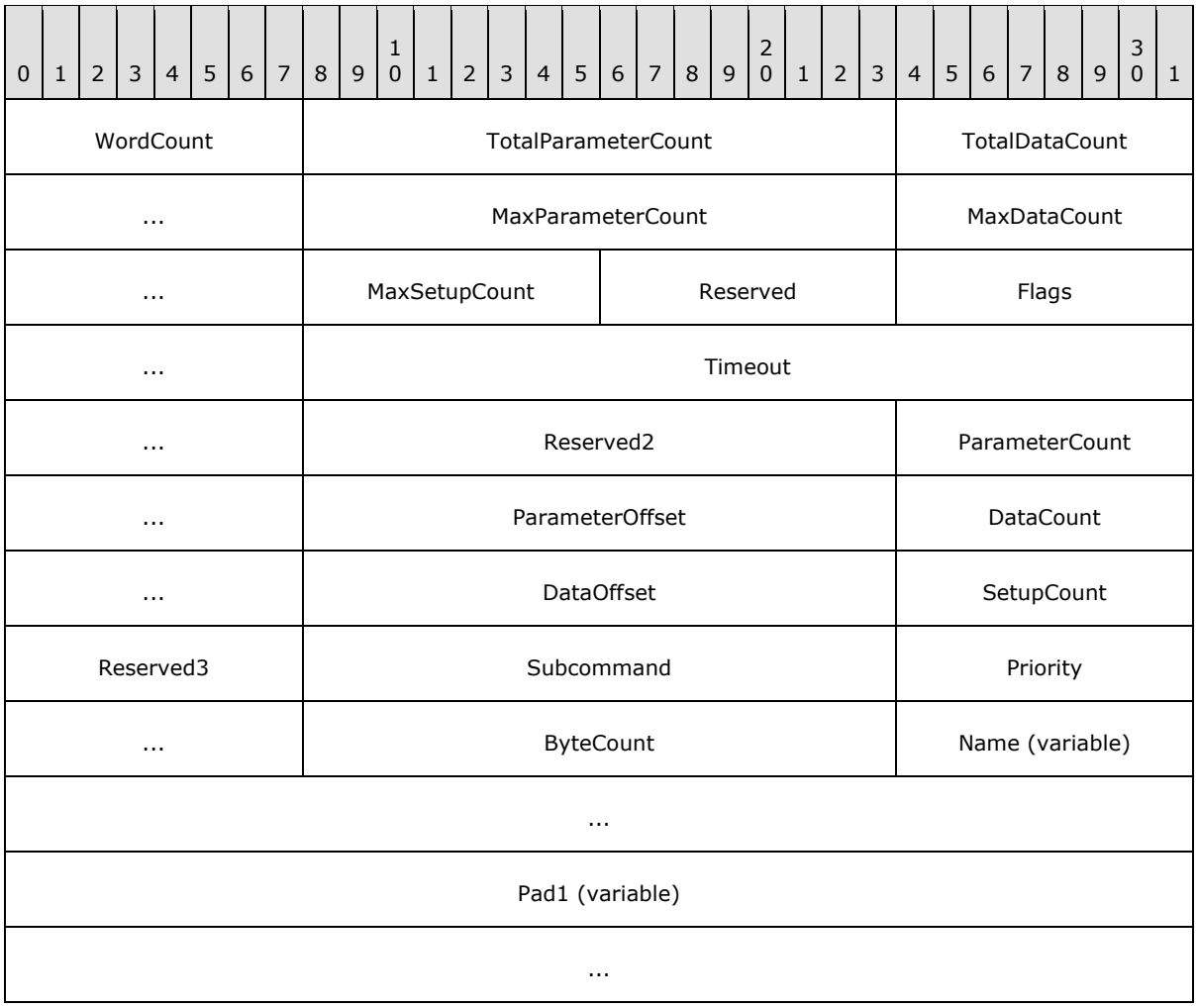
ByteCount (2 bytes): This value MUST be set to the number of bytes that follows the **ByteCount** field in this packet.

Pad1 (1 byte): A potential padding to set the start of the Parameter buffer that MUST be on a 4-byte boundary from the start of the SMB header.

2.2.12.19 TRANS_WAIT_NMPIPE Request

This TRANS_WAIT_NMPIPE subcommand of SMB_COM_TRANSACTION allows a client to be notified when the specified named pipe is available to be connected to.

The format follows the convention for SMB_COM_TRANSACTION, as specified in [CIFS] section 3.13.1.



WordCount (1 byte): The count of 16-bit words in the request structure. For this request, it MUST be 0x10 (16), which is 14 plus the **SetupCount** value of 2.

TotalParameterCount (2 bytes): The number of bytes in the Parameter buffer. For this request, it MUST be 0. The content of the Parameter buffer is described below.

TotalDataCount (2 bytes): The number of bytes in the Data buffer. For this request, it MUST be set to 0.

MaxParameterCount (2 bytes): The maximum number of bytes the client will accept in the Parameter buffer of the response. For this request, it MUST be set to 0.

MaxDataCount (2 bytes): The maximum number of bytes the client will accept in the Data buffer of the response. For this request, it MUST be set to 0.

MaxSetupCount (1 byte): The maximum number of bytes the client will accept in the Setup buffer of the response. For this request, it MUST be set to 0.

Reserved (1 byte): An unused value that SHOULD be set to 0 when sending this message. The server MUST ignore this field when receiving this message.

Flags (2 bytes): Set of options that provide additional information to the server on this request. The client MAY set either of the following bits. Unused bit fields SHOULD be set to 0 by the client when sending a request, and MUST be ignored when received by the server.

Value	Meaning
DISCONNECT_TID 0x0001	Server MUST disconnect the tree connect associated with the Tid received in the SMB header of this request after the request completes. The client SHOULD NOT send an SMB_COM_TREE_DISCONNECT for this tree connect.
NO_RESPONSE 0x0002	Server MUST process this client request as a one-way transaction and MUST NOT send a response back to the client.<108>

Timeout (4 bytes): The maximum amount of time in milliseconds to wait for the operation to complete. The client MAY set this to 0 to indicate that no time-out is given. If the operation does not complete within the specified time, the server MAY abort the request and send a failure response.<109>

Reserved2 (2 bytes): An unused value that SHOULD be set to 0 when sending this message. The server MUST ignore this field when receiving this message.

ParameterCount (2 bytes): The count of bytes in the Parameter buffer of this packet. For this request, it MUST be 0.

ParameterOffset (2 bytes): An offset in bytes from the start of the SMB header for this request to the start of the Parameter buffer. The Parameter buffer MUST start after the Name field. The Parameter buffer MUST be aligned to start on a 4-byte boundary from the start of the SMB header.

DataCount (2 bytes): The count of bytes in the Data buffer field. For this request, it MUST be set to 0.

DataOffset (2 bytes): An offset in bytes to the start of Data buffer field from the start of the SMB header for this request. This value MUST be greater than or equal to **ParameterOffset** plus **ParameterCount**. The Data buffer MUST be aligned to start on a 4-byte boundary from the start of the SMB header. If the **DataCount** field is 0, the client MAY set the **DataOffset** field to 0, or the client MAY set this field to a location in the packet after where the Parameter bytes would be placed, if present.<110>

SetupCount (1 byte): The number of setup words. For this request, it MUST be set to 2.

Reserved3 (1 byte): An unused value that SHOULD be set to 0 when sending this message. The server MUST ignore this field when receiving this message.

Subcommand (2 bytes): The first setup word for this request. This field MUST be set to TRANS_WAIT_NMPIPE, the subcommand to request. For a list of named pipe subcommands and values, see section 2.2.12.

Priority (2 bytes): The second setup word, which indicates the numeric priority of the request for the named pipe, MUST be in the range of 0 to 9. The larger the value, the higher the priority.<111>

ByteCount (2 bytes): This value MUST be the number of bytes that follows the **ByteCount** field in this packet.

Name (variable): The name field MUST be set to the name of the pipe being waited on, in the format \PIPE\<pipename> where <pipename> is the name of the pipe to wait to connect to.

To wait on the pipe PipeA, the name field is set to \PIPE\PipeA. If SMB_FLAGS2_UNICODE is set in the **Flags2** field of the SMB header of the request, the name string MUST be a null-terminated array of 16-bit Unicode characters. Otherwise, the name string MUST be a null-terminated array of OEM characters. A client MAY set this to the named pipe name. The server MUST ignore this field in the request. If the name string consists of Unicode characters, this field MUST be aligned to start on a 2-byte boundary from the start of the SMB header.

Pad1 (variable): An optional padding to set the Parameter buffer to start on a 4-byte boundary from the start of the SMB header.

2.2.12.20 TRANS_WAIT_NMPIPE Response

The server MUST send a TRANS_WAIT_NMPIPE response in reply to a client TRANS_WAIT_NMPIPE subcommand request when the request is successful. The server MUST set an error code in the **Status** field of the SMB header of the response to indicate whether the transact operation on the named pipe was successful or failed. The server returns a response when either the named pipe is available to be connected to or the **Timeout** field specified in the client request has been exceeded. If the **Timeout** value is exceeded, the server returns STATUS_IO_TIMEOUT in the **Status** field of the SMB header. If the named pipe is available to be connected to, and the **Timeout** is not exceeded, the server returns STATUS_SUCCESS in the **Status** field of the SMB header.

The format follows the convention for SMB_COM_TRANSACTION, as specified in [CIFS] section 3.13.1.

0	1	2	3	4	5	6	7	8	9	10	1	2	3	4	5	6	7	8	9	20	1	2	3	4	5	6	7	8	9	30	1
WordCount								TotalParameterCount																TotalDataCount							
...								Reserved																ParameterCount							
...								ParameterOffset																ParameterDisplacement							
...								DataCount																DataOffset							
...								DataDisplacement																SetupCount							
Reserved2								ByteCount																Pad1 (optional)							

WordCount (1 byte): The count of 16-bit words in the response structure. For this response, this MUST be set to 0x0A (10), which is 10 plus the **SetupCount** of 0.

TotalParameterCount (2 bytes): The number of bytes in the Parameter buffer. For this response, it MUST be set to 0.

TotalDataCount (2 bytes): The number of bytes in the Data buffer. For this response, it MUST be set to 0.

Reserved (2 bytes): An unused value that SHOULD be set to 0 when sending this response. The client MUST ignore this field when receiving this message.

ParameterCount (2 bytes): The number of bytes in the Parameter buffer contained in this packet. For this response, it MUST be set to 0.

ParameterOffset (2 bytes): An offset in bytes from the start of the SMB header to the start of the part of the Parameter buffer contained in this response. The Parameter buffer MUST start after the **Reserved2** field. The Parameter buffer MUST be aligned to start on a 4-byte boundary from the start of the SMB header. If the **ParameterCount** field is 0, the server MAY set the ParameterOffset field to 0, or the server MAY set this field to a location in the packet after where Parameter bytes would have been placed, if present.<112>

ParameterDisplacement (2 bytes): An offset in bytes into the final Parameters buffer assembled from all responses. For a single buffer transaction (whose Parameters buffer fits in a single response), this value MUST be set to 0.

DataCount (2 bytes): The number of bytes in the Data buffer contained in this packet. For this response, it MUST be set to 0.

DataOffset (2 bytes): An offset in bytes from the start of the SMB header to the start of the part of the Data buffer contained in this response. This value MUST be greater than ParameterOffset plus **ParameterCount**. The Data buffer MUST be aligned to start on a 4-byte boundary from the start of the SMB header. If the **DataCount** field is 0, the server MAY set the **DataOffset** field to 0, or the server MAY set this field to a location in the packet after where the Parameter bytes would be placed, if present.<113>

DataDisplacement (2 bytes): An offset in bytes into the final Data buffer assembled from all responses. For a single buffer transaction (whose Data buffer fits in a single response), this value MUST be set to 0.

SetupCount (1 byte): The number of setup words. For this request, it MUST be set to 0.

Reserved2 (1 byte): An unused value that SHOULD be set to 0 when sending this response. The client MUST ignore this field when receiving this message.

ByteCount (2 bytes): This value MUST be set to the number of bytes that follows the **ByteCount** field in this packet.

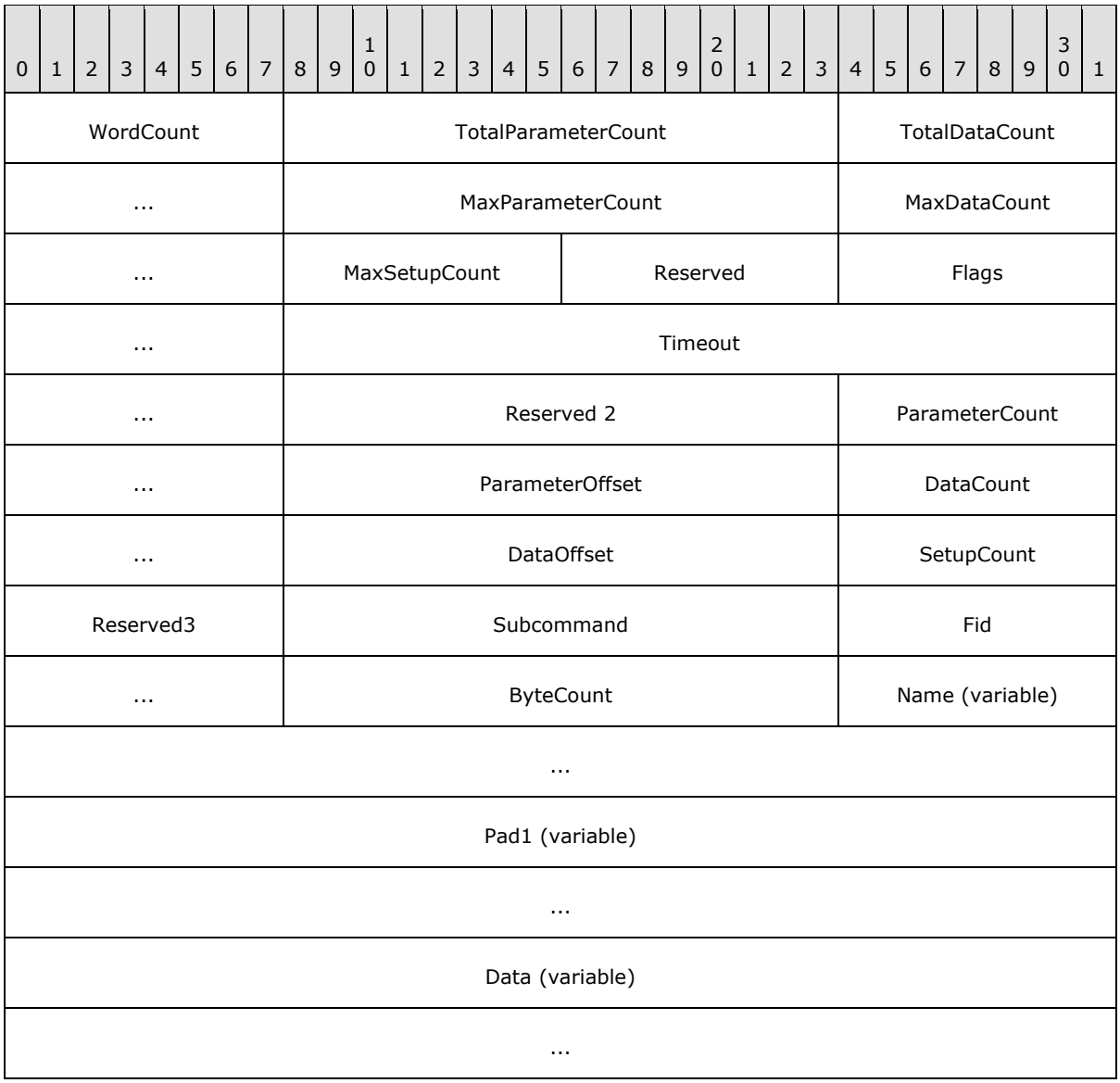
Pad1 (1 byte): A potential padding to set the start of the Parameter buffer that MUST be set on a 4-byte boundary from the start of the SMB header.

2.2.12.21 TRANS_CALL_NMPIPE Request

The TRANS_CALL_NMPIPE subcommand allows a client to connect to a named pipe, issue a write to the named pipe, issue a read from the named pipe, and close the named pipe.

The format follows the convention for SMB_COM_TRANSACTION, as specified in [CIFS] section 3.13.1.

The TransactionName MUST contain the name of the pipe being connected to.**TotalDataCount** MUST be the number of bytes being written to the pipe, with that data contained in the Data buffer. **MaxDataCount** MUST be the number of bytes to be read from the pipe.<114>



WordCount (1 byte): The count of 16-bit words in the request structure. For this request, it MUST be 0x10 (16), which is 14 plus the SetupCount of 2.

TotalParameterCount (2 bytes): The number of bytes in the Parameter buffer. For this request, it MUST be 2. The content of the Parameter buffer is described below.

TotalDataCount (2 bytes): The number of bytes in the Data buffer. For this request, it MUST be set to the number of bytes to write to the named pipe.

MaxParameterCount (2 bytes): The maximum number of bytes the client will accept in the Parameter buffer of the response. For this request, it MUST be set to 0.

MaxDataCount (2 bytes): The maximum number of bytes the client will accept in the Data buffer of the response. For this request, it MUST be set to the number of bytes to be read from the named pipe.

MaxSetupCount (1 byte): The maximum number of bytes the client will accept in the Setup buffer of the response. For this request, it MUST be set to 0.

Reserved (1 byte): An unused value that SHOULD be set to 0 when sending this message. The server MUST ignore this field when receiving this message.

Flags (2 bytes): A set of options that provide additional information to the server on this request. The client MAY set either of the following bits. Unused bit fields SHOULD be set to 0 by the client when sending a request, and MUST be ignored when received by the server.

Value	Meaning
DISCONNECT_TID 0x0001	The server MUST disconnect the tree connect associated with the Tid received in the SMB header of this request after the request completes. The client SHOULD NOT send an SMB_COM_TREE_DISCONNECT for this tree connect.
NO_RESPONSE 0x0002	The server MUST process this client request as a one-way transaction and MUST NOT send a response back to the client.<115>

Timeout (4 bytes): MUST be the maximum amount of time in milliseconds to wait for the operation to complete. The client MAY set this to 0 to indicate that no time-out is given. If the operation does not complete within the specified time, the server MAY abort the request and send a failure response.<116>

Reserved 2 (2 bytes): An unused value that SHOULD be set to 0 when sending this message. The server MUST ignore this field when receiving this message.

ParameterCount (2 bytes): The count of bytes in the Parameter buffer of this packet. For this request, it MUST be set to 2.

ParameterOffset (2 bytes): An offset in bytes from the start of the SMB header for this request to the start of the Parameter buffer. The Parameter buffer MUST start after the Name field. The Parameter buffer MUST be aligned to start on a 4-byte boundary from the start of the SMB header.

DataCount (2 bytes): The count of bytes in the Data buffer field. For this request, it MUST be set to the number of bytes to write to the named pipe in this packet.

DataOffset (2 bytes): An offset in bytes to the start of Data buffer field from the start of the SMB header for this request. This value MUST be greater than or equal to **ParameterOffset** plus **ParameterCount**. The Data buffer MUST be aligned to start on a 4-byte boundary from the start of the SMB header.

SetupCount (1 byte): The number of setup words. For this request, it MUST be set to 2.

Reserved3 (1 byte): An unused value that SHOULD be set to 0 when sending this message. The server MUST ignore this field when receiving this message.

Subcommand (2 bytes): The first setup word for this request. This field MUST be set to TRANS_CALL_NMPIPE, the subcommand to request. For a list of named pipe subcommands and values, see 2.2.12.

Fid (2 bytes): The second setup word for this request. This field is the SMB file identifier for the named pipe that is to be connected to and called. This field MUST be set to a valid Fid from a server response for a previous SMB command to open or create a named pipe. These commands include SMB_COM_OPEN, SMB_COM_CREATE, SMB_COM_CREATE_TEMPORARY,

SMB_COM_CREATE_NEW, SMB_COM_OPEN_ANDX, SMB_COM_NT_CREATE_ANDX, and SMB_COM_NT_TRANSACT with subcommand NT_TRANSACT_CREATE.

ByteCount (2 bytes): This value MUST be the number of bytes that follows the **ByteCount** field in this packet.

Name (variable): An optional name for this transaction. If SMB_FLAGS2_UNICODE is set in the **Flags2** field of the SMB header of the request, the name string MUST be a null-terminated array of 16-bit Unicode characters. Otherwise, the name string MUST be a null-terminated array of OEM characters. A client MAY set this to the named pipe name. The server MUST ignore this field in the request. If the name string consists of Unicode characters, this field MUST be aligned to start on a 2-byte boundary from the start of the SMB header.<117>

Pad1 (variable): An optional padding to set the Parameter buffer (the **PipeState** field) that MUST start on a 4-byte boundary from the start of the SMB header.

Data (variable): Data buffer that contains the data to write to the named pipe.

2.2.12.22 TRANS_CALL_NMPIPE Response

The server MUST send a TRANS_CALL_NMPIPE subcommand response in reply to a client TRANS_CALL_NMPIPE request when the request is successful. The server MUST set an error code in the Status field of the SMB header of the response to indicate whether the call operation on the named pipe was successful or failed.

The format follows the convention for SMB_COM_TRANSACTION, as specified in [CIFS] section 3.13.1.

0	1	2	3	4	5	6	7	8	9	10	1	2	3	4	5	6	7	8	9	20	1	2	3	4	5	6	7	8	9	30	1
WordCount								TotalParameterCount																TotalDataCount							
...								Reserved																ParameterCount							
...								ParameterOffset																ParameterDisplacement							
...								DataCount																DataOffset							
...								DataDisplacement																SetupCount							
Reserved 2								ByteCount																Pad1 (optional)							
Data (variable)																															
...																															

WordCount (1 byte): The count of 16-bit words in the response structure. For this response, this MUST be set to 0x0A (10), which is 10 plus the SetupCount of 0.

TotalParameterCount (2 bytes): The number of bytes in the Parameter buffer. For this response, it MUST be set to 0.

TotalDataCount (2 bytes): The number of bytes in the Data buffer. For this response, it MUST be set to the number of bytes read from the pipe.

Reserved (2 bytes): An unused value that SHOULD be set to 0 when sending this response. The client MUST ignore this field when receiving this message.

ParameterCount (2 bytes): The number of bytes in the Parameter buffer contained in this packet. For this response, it MUST be set to 0.

ParameterOffset (2 bytes): An offset in bytes from the start of the SMB header to the start of the part of the Parameter buffer contained in this response. The Parameter buffer MUST start after the **Reserved2** field. The Parameter buffer MUST be aligned to start on a 4-byte boundary from the start of the SMB header. If the **ParameterCount** field is 0, the server MAY set the **ParameterOffset** field to 0, or the server MAY set this field to a location in the packet after where Parameter bytes would have been placed, if present.<118>

ParameterDisplacement (2 bytes): An offset in bytes into the final Parameters buffer assembled from all responses. For a single buffer transaction (whose Parameters buffer fits in a single response), this value MUST be set to 0.

DataCount (2 bytes): The number of bytes in the Data buffer contained in this packet. For this response, it MUST be set to the number of bytes read from the pipe in this packet.

DataOffset (2 bytes): An offset in bytes from the start of the SMB header to the start of the part of the Data buffer contained in this response. This value MUST be greater than **ParameterOffset** plus **ParameterCount**. The Data buffer MUST be aligned to start on a 4-byte boundary from the start of the SMB header. If the **DataCount** field is 0, the server MAY set the **DataOffset** field to 0, or the server MAY set this field to a location in the packet after where the Parameter bytes would be placed, if present.<119>

DataDisplacement (2 bytes): An offset in bytes into the final Data buffer assembled from all responses. For a single buffer transaction (whose Data buffer fits in a single response), this value MUST be set to 0.

SetupCount (1 byte): The number of setup words. For this request, it MUST be set to 0.

Reserved 2 (1 byte): An unused value that SHOULD be set to 0 when sending this response. The client MUST ignore this field when receiving this message.

ByteCount (2 bytes): This value MUST be set to the number of bytes that follows the **ByteCount** field in this packet.

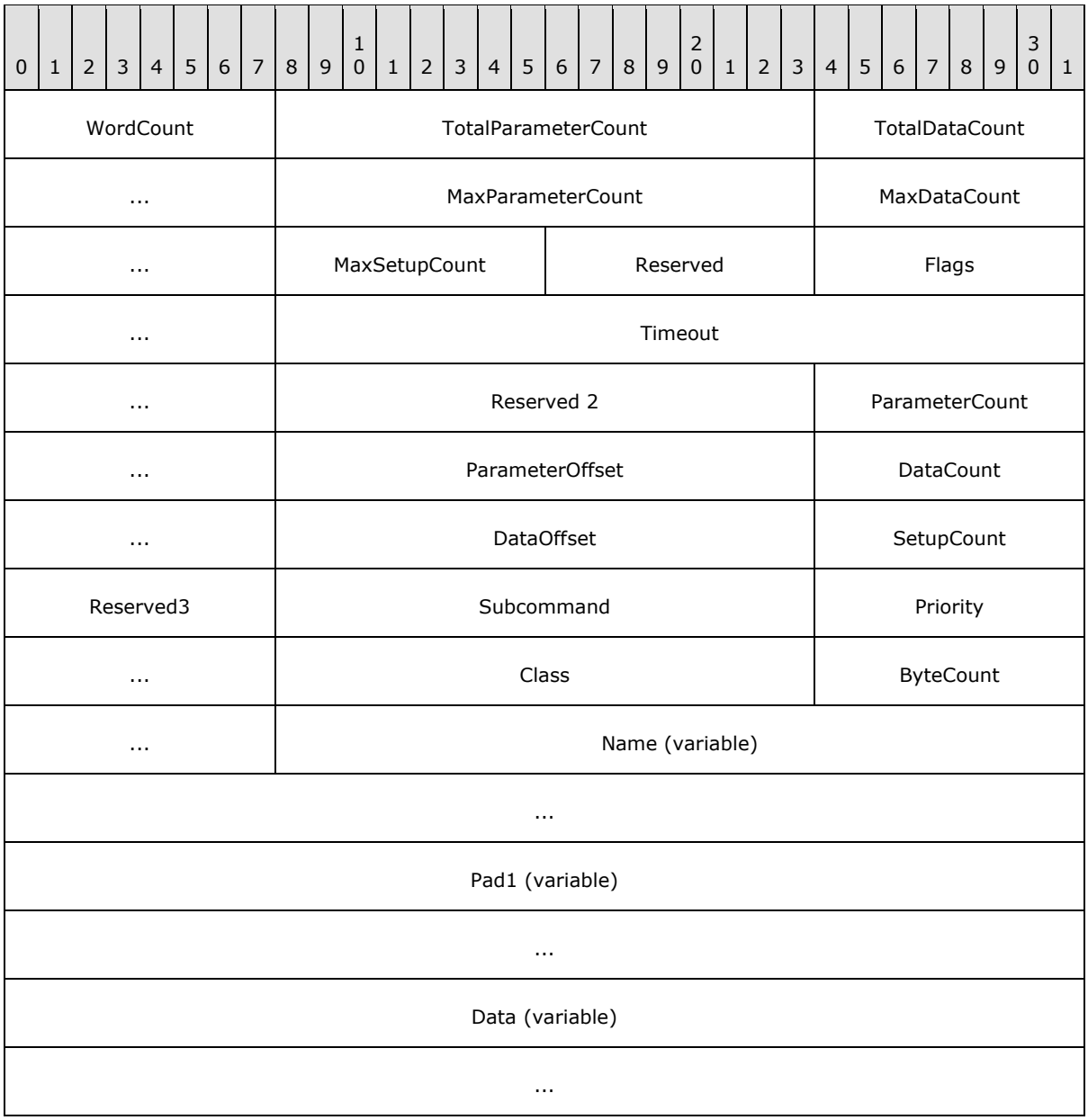
Pad1 (1 byte): A potential padding to set the start of the Parameter buffer that MUST be set on a 4-byte boundary from the start of the SMB header.

Data (variable): The Data buffer that contains the data read from the named pipe.

2.2.12.23 TRANS_MAILSLOT_WRITE Request

This TRANS_MAILSLOT_WRITE subcommand allows a client to write data to a specific mailslot on the server.<120>

The format follows the convention for SMB_COM_TRANSACTION, as specified in [CIFS] section 3.13.1.



WordCount (1 byte): The count of 16-bit words in the request structure. For this request, it MUST be 0x11 (17), which is 14 plus the SetupCount of 3.

TotalParameterCount (2 bytes): The number of bytes in the Parameter buffer. For this request, it MUST be 0.

TotalDataCount (2 bytes): The number of bytes in the Data buffer. For this request, it MUST be the number of bytes to write to the mailslot.

MaxParameterCount (2 bytes): The maximum number of bytes the client will accept in the Parameter buffer of the response. For this request, it MUST be set to 2.

MaxDataCount (2 bytes): The maximum number of bytes the client will accept in the Data buffer of the response. For this request, it MUST be 0.

MaxSetupCount (1 byte): The maximum number of bytes the client will accept in the Setup buffer of the response. For this request, it MUST be 0.

Reserved (1 byte): An unused value that SHOULD be set to 0 when sending this message. The server MUST ignore this field when receiving this message.

Flags (2 bytes): A set of options that provide additional information to the server on this request. The client MAY set either of the following bits. Unused bit fields SHOULD be set to 0 by the client when sending a request, and MUST be ignored when received by the server.

Value	Meaning
DISCONNECT_TID 0x0001	The server MUST disconnect the tree connect associated with the Tid received in the SMB header of this request after the request completes. The client SHOULD NOT send an SMB_COM_TREE_DISCONNECT for this tree connect.
NO_RESPONSE 0x0002	The server MUST process this client request as a one-way transaction and MUST NOT send a response back to the client.<121>

Timeout (4 bytes): The maximum amount of time in milliseconds to wait for the operation to complete. The client MAY set this to 0 to indicate that no time-out is given. If the operation does not complete within the specified time, the server MAY abort the request and send a failure response.<122>

Reserved 2 (2 bytes): An unused value that SHOULD be set to 0 when sending this message. The server MUST ignore this field when receiving this message.

ParameterCount (2 bytes): The count of bytes in the Parameter buffer of this packet. For this request, it MUST be set to 0.

ParameterOffset (2 bytes): An offset in bytes from the start of the SMB header for this request to the start of the Parameter buffer. The Parameter buffer MUST start after the **Name** field. The Parameter buffer MUST be aligned to start on a 4-byte boundary from the start of the SMB header. If the **ParameterCount** field is 0, the client MAY set the **ParameterOffset** field to 0, or the client MAY set this field to a location in the packet after the Name field where Parameter bytes would have been placed, if present.

DataCount (2 bytes): The number of bytes in the Data buffer field. For this request, it MUST be set to the number of bytes in this packet to write to the mailslot.

DataOffset (2 bytes): An offset in bytes to the start of Data buffer field from the start of the SMB header for this request. This value MUST be greater than or equal to **ParameterOffset** plus **ParameterCount**. The Data buffer MUST be aligned to start on a 4-byte boundary from the start of the SMB header.

SetupCount (1 byte): The number of setup words. For this request, it MUST be set to 3.

Reserved3 (1 byte): An unused value that SHOULD be set to 0 when sending this message. The server MUST ignore this field when receiving this message.

Subcommand (2 bytes): The first setup word for this request. This field MUST be set to TRANS_MAILSLOT_WRITE, the subcommand to request. For a list of mailslot subcommands and values, see 2.2.12.

Priority (2 bytes): The second setup word, the numeric priority of the message being written to the mailslot, MUST be in the range of 0 to 9. The larger the value, the higher the priority.<123>

Class (2 bytes): The third setup word, the class of the mailslot request.<124> This value MUST be set to one of the following values.

Mailslot class/Value	Meaning
Class 1 0x0001	A first-class mailslot is reliable and guarantees the delivery of the message. This class MAY transmit messages of up to 65,535 bytes. Messages to class 1 mailslots MUST not be broadcast.
Class 2 0x0002	A second-class mailslot is unreliable and does not guarantee delivery. This class MAY transmit messages up to a maximum length that depends on the configuration of the server, but will never be less than 360 bytes. Messages to class 2 mailslot MAY be broadcast, which allows a message to be sent to a particular mailslot on all systems.

ByteCount (2 bytes): This value MUST be the number of bytes that follows the **ByteCount** field in this packet.

Name (variable): Name of the mailslot being written to. The name string MUST be a null-terminated array of ASCII characters in the format \MAILSLOT\<mailslotname> where <mailslotname> is the name of the MAILSLOT.

Pad1 (variable): An optional padding to set the Parameter buffer (the **PipeState** field) that MUST start on a 4-byte boundary from the start of the SMB header.

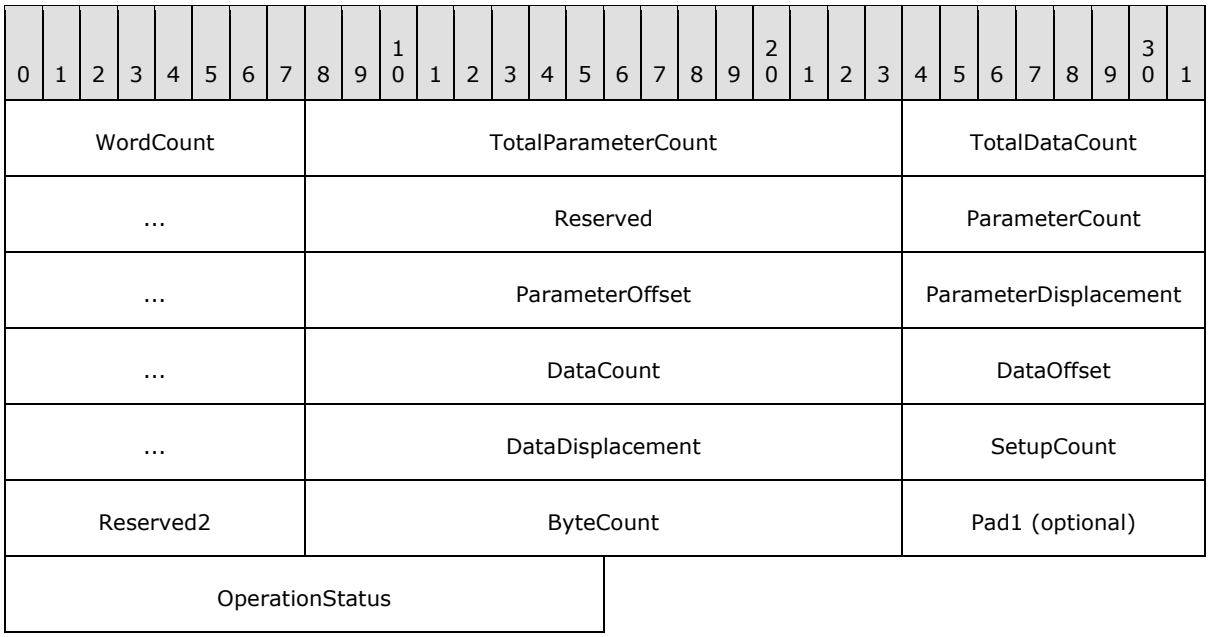
Data (variable): Data buffer that contains the data to write to the mailslot.

2.2.12.24 TRANS_MAILSLOT_WRITE Response

The server MUST send a TRANS_MAILSLOT_WRITE response in reply to a client TRANS_MAILSLOT_WRITE (section 2.2.12.23) request when the request is successful. The server MUST set an error code in the **Status** field of SMB header of the response to indicate whether the transact operation on the named pipe was successful or failed.<125>

The format follows the convention for SMB_COM_TRANSACTION, as specified in [CIFS] section 3.13.1.

TotalDataCount MUST be 0. **TotalParameterCount** MUST be 2 with a 16-bit status value being passed in the Parameter block. If the connection does not support 32-bit status codes, as specified in [CIFS] section 3.1.6, the status value in the Parameter block MUST be set to the same value as the **Status.DosError.Error** field of the SMB header. If the connection does support 32-bit status codes, the status value in the Parameter block MUST be set to 0xFFFF.



WordCount (1 byte): The count of 16-bit words in the response structure. For this response, this MUST be set to 0x0A (10), which is 10 plus the SetupCount of 0.

TotalParameterCount (2 bytes): The number of bytes in the Parameter buffer. For this response, it MUST be set to 2.

TotalDataCount (2 bytes): The number of bytes in the Data buffer. For this response, it MUST be set to 0.

Reserved (2 bytes): An unused value that SHOULD be set to 0 when sending this response. The client MUST ignore this field when receiving this message.

ParameterCount (2 bytes): The number of bytes in the Parameter buffer contained in this packet. For this response, it MUST be set to 2.

ParameterOffset (2 bytes): An offset in bytes from the start of the SMB header to the start of the part of the Parameter buffer contained in this response. The Parameter buffer MUST start after the **Reserved2** field. The Parameter buffer MUST be aligned to start on a 4-byte boundary from the start of the SMB header.

ParameterDisplacement (2 bytes): An offset in bytes into the final Parameters buffer assembled from all responses. For a single buffer transaction (whose Parameters buffer fits in a single response), this value MUST be set to 0.

DataCount (2 bytes): The number of bytes in the Data buffer contained in this packet. For this response, it MUST be set to 0.

DataOffset (2 bytes): An offset in bytes from the start of the SMB header to the start of the part of the Data buffer contained in this response. This value MUST be greater than **ParameterOffset** plus **ParameterCount**. The Data buffer MUST be aligned to start on a 4-byte boundary from the start of the SMB header. If the **DataCount** field is 0, the server MAY set the **DataOffset** field to 0, or the server MAY set this field to a location in the packet after where the Parameter bytes would be placed, if present.<126>

DataDisplacement (2 bytes): An offset in bytes into the final Data buffer assembled from all responses. For a single buffer transaction (whose Data buffer fits in a single response), this value MUST be set to 0.

SetupCount (1 byte): The number of setup words. For this request, it MUST be set to 0.

Reserved2 (1 byte): An unused value that SHOULD be set to 0 when sending this response. The client MUST ignore this field when receiving this message.

ByteCount (2 bytes): This value MUST be set to the number of bytes that follows the **ByteCount** field in this packet.

Pad1 (1 byte): A potential padding to set the start of the Parameter buffer that MUST be set on a 4-byte boundary from the start of the SMB header.

OperationStatus (2 bytes): The Parameter buffer contains a single USHORT (as specified in [MS-DTYP] section **2.2.57**). If the SMB connection does not support 32-bit status codes (see the SMB_COM_NEGOTIATE server response in SMB_COM_NEGOTIATE Server Response Extension (section 2.2.3) and the SMB_COM_SESSION_SETUP_ANDX client request in SMB_COM_SESSION_SETUP_ANDX Client Request Extension (section 2.2.4)) or the TRANS_MAILSLOT_WRITE client request did not have the SMB_FLAGS2_NT_STATUS bit set in the **Flags2** field of the SMB header, **OperationStatus** MUST be set to the same value as the **Status.DosError.Error** field of the SMB header of the response. If the connection does support 32-bit status codes, **OperationStatus** MUST be set to 0xffff.<127>

2.2.13 SMB_COM_TRANSACTION2 Extensions

The SMB_COM_TRANSACTION2 request is sent by a client to execute a specific operation on the server including file enumeration, query and set file attribute operations, and DFS referral retrieval. The possible subcommand codes in the SMB_COM_TRANSACTION2 requests and responses are as specified in [CIFS] section 6.2, and the execution of these commands is as specified in [CIFS] section 3.13.1. Changes to these subcommands are defined below.

A new constant SMB_INFO_PASSTHROUGH is defined as 0x03e8 (decimal 1,000). New values for the **InformationLevel** field can be set in a client request for a number of subcommands of the SMB_COM_TRANSACTION2. An **InformationLevel** value of SMB_INFO_PASSTHROUGH and higher indicates the client is requesting a native Windows NT information level, as specified in section 3.2.4.7 and in [MS-FSCC] section 2.4. The client requests a native Windows NT information level by adding SMB_INFO_PASSTHROUGH to the value of the native Windows NT information level, and then passes this in the **InformationLevel** field in the client request. If the server supports the CAP_INFOLEVEL_PASSTHRU (the CAP_INFOLEVEL_PASSTHRU bit is set in the **Capabilities** field in the SMB_COM_NEGOTIATE response), the Data buffer in the server response is formatted with information based on the native Windows NT information level requested. The specific format of the data block in the server response is based on the native Windows NT information level, as specified in [MS-FSCC] section 2.4.

The command being executed by a transaction is determined by the **SetupCount** fields in the SMB_COM_TRANSACTION2 request, as specified in [CIFS] section 3.13.1. The first Setup WORD MUST be a subcommand code to execute. The parameters associated with any particular request are placed in the Parameters buffer of the request. If the server receives a request with an unknown subcommand code, it MUST fail the request with STATUS_INVALID_PARAMETER. One new subcommand, TRANS2_SET_FS_INFORMATION (section 2.2.13.11), is added in these extensions. For completeness, the following is an inclusive list of subcommands.

Name of subcommand code	Value
TRANS2_OPEN2	0x0000
TRANS2_FIND_FIRST2	0x0001
TRANS2_FIND_NEXT2	0x0002
TRANS2_QUERY_FS_INFORMATION	0x0003
TRANS2_SET_FS_INFORMATION	0x0004
TRANS2_QUERY_PATH_INFORMATION	0x0005
TRANS2_SET_PATH_INFORMATION	0x0006
TRANS2_QUERY_FILE_INFORMATION	0x0007
TRANS2_SET_FILE_INFORMATION	0x0008
TRANS2_FSCTL	0x0009
TRANS2_IOCTL2	0x000a
TRANS2_FIND_NOTIFY_FIRST	0x000b
TRANS2_FIND_NOTIFY_NEXT	0x000c
TRANS2_CREATE_DIRECTORY	0x000d
TRANS2_SESSION_SETUP	0x000e
TRANS2_GET_DFS_REFERRAL	0x0010
TRANS2_REPORT_DFS_INCONSISTENCY	0x0011

Client requests that require sending more data than the negotiated maximum server buffer size (the **MaxBufferSize** field in the SMB_COM_NEGOTIATE response) MUST be split into multiple requests. The initial primary request MUST be sent as a SMB_COM_TRANSACTION2 primary client request. The initial primary request MUST include the **SetupCount** bytes and the **Name** field for a transaction to be successful. The subsequent secondary requests MUST be sent as multiple SMB_COM_TRANSACTION2_SECONDARY secondary client requests. The following subcommand client requests MAY require sending more data than the negotiated server maximum buffer size: TRANS2_SET_FS_INFORMATION, TRANS2_SET_FILE_INFORMATION, TRANS2_SET_PATH_INFORMATION, TRANS2_FSCTL, and TRANS2_IOCTL2.

Server responses that require sending more data than the negotiated maximum client buffer size (the **MaxBufferSize** field in the SMB_COM_SESSION_SETUP request) MUST be sent as multiple SMB_COM_TRANSACTION2 interim server responses followed by a single final SMB_COM_TRANSACTION2 server response. The following subcommand server responses MAY require sending more data than the negotiated maximum client buffer size: TRANS2_FIND_FIRST2, TRANS2_FIND_NEXT2, TRANS2_QUERY_FS_INFORMATION, TRANS2_QUERY_FILE_INFORMATION, TRANS2_QUERY_PATH_INFORMATION, TRANS2_FSCTL, TRANS2_IOCTL2, TRANS2_FIND_NOTIFY_FIRST, TRANS2_FIND_NOTIFY_NEXT, and TRANS2_GET_DFS_REFERRAL.

2.2.13.1 TRANS2_QUERY_FILE_INFORMATION Request

A TRANS2_QUERY_FILE_INFORMATION subcommand of the SMB_COM_TRANSACTION2 is sent by a client to request attribute information for a file or directory that has been opened, as specified in [CIFS] section 4.2.15.

This extension provides a new **InformationLevel** value range that can be used to query information from the server. **InformationLevel** values that are SMB_INFO_PASSTHROUGH (as specified in section 2.2.13) or higher indicate that the client is requesting a native Windows NT information level. The native Windows NT information levels are specified in section 3.2.4.7 and in [MS-FSCC] section 2.2.4.

2.2.13.2 TRANS2_QUERY_FILE_INFORMATION Response

A server MUST send a TRANS2_QUERY_FILE_INFORMATION response in reply to a SMB_COM_TRANSACTION2 client request with a TRANS2_QUERY_FILE_INFORMATION subcommand when the request is successful. The Data block of the transaction response contains the information requested, as specified in [CIFS] section 4.2.15.

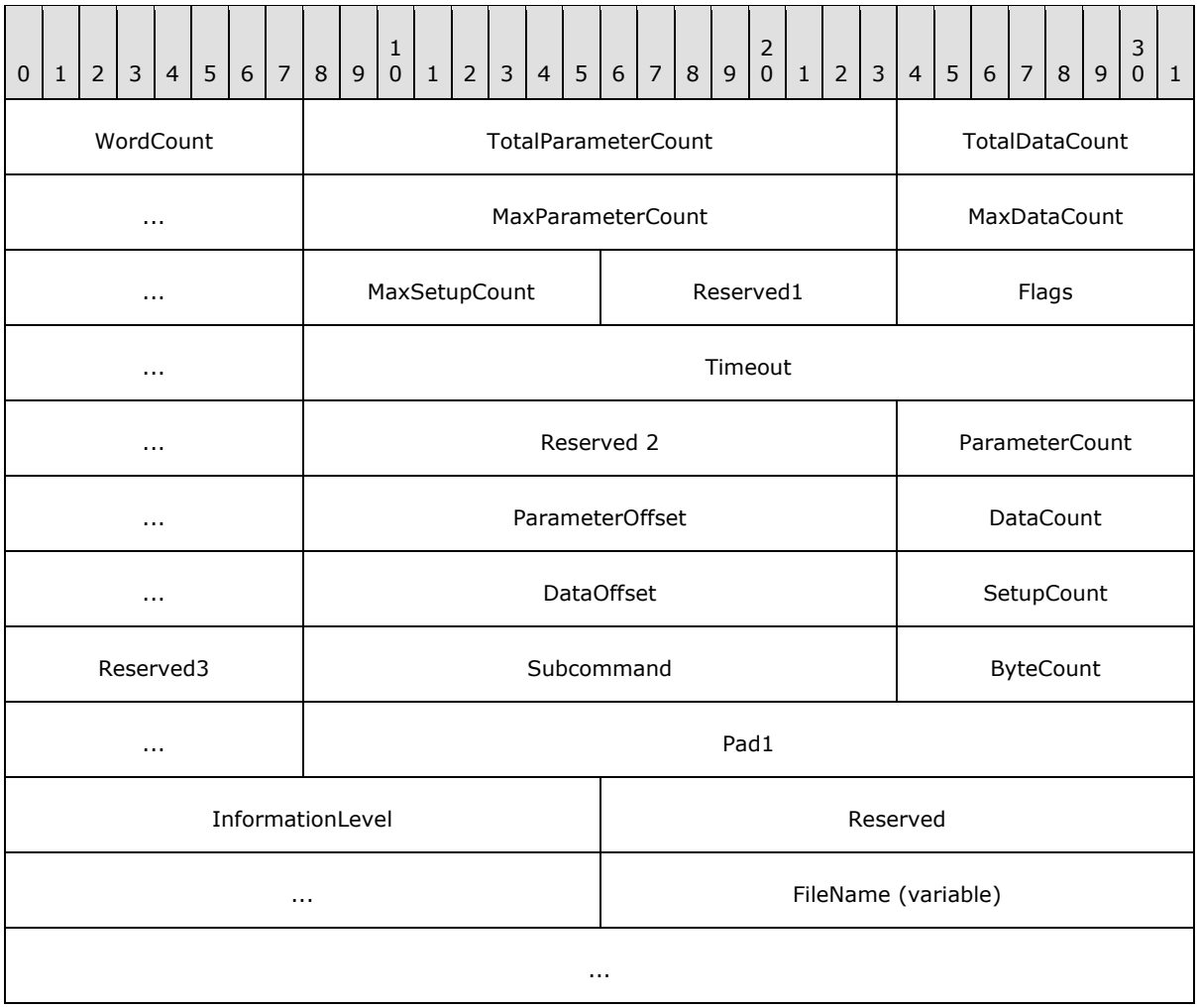
If the **InformationLevel** field in the client request was SMB_INFO_PASSTHROUGH (as specified in section 2.2.13) and higher, the Data block is formatted with a native Windows NT-specific format. For more information, see section 3.2.4.7 and [MS-FSCC] section 2.4.

2.2.13.3 TRANS2_QUERY_PATH_INFORMATION Request

A TRANS2_QUERY_PATH_INFORMATION subcommand of the SMB_COM_TRANSACTION2 is sent by the client to request attribute information for a file or directory by specifying the path, as specified in [CIFS] section 4.2.14.

The client MUST NOT send this subcommand if the server does not support NT SMBs as specified by the CAP_NT_SMBS bit of the **Capabilities** field of the SMB_COM_NEGOTIATE server response (section 2.2.3).

This extension provides a new **InformationLevel** value range that can be used to set information on the server. **InformationLevel** values that are SMB_INFO_PASSTHROUGH (as specified in section 2.2.13) or higher indicate that the client is requesting the use of a native Windows NT information level.



WordCount (1 byte): A count of 16-bit words in the request structure. For this request, it MUST be 0x0F (15).

TotalParameterCount (2 bytes): The number of bytes in the Parameter buffer. The content of the Parameter buffer is described below.

TotalDataCount (2 bytes): The number of bytes in the Data buffer. For this request, it MUST be set to 0.

MaxParameterCount (2 bytes): The maximum number of bytes the client will accept in the Parameter buffer of the response. For this request, it MUST be set to 2.

MaxDataCount (2 bytes): The maximum number of bytes the client will accept in the Data buffer of the response. For this request, it MUST be set to a value greater than the size of the Windows NT-specific structure associated with a particular information level.

MaxSetupCount (1 byte): The maximum number of bytes the client will accept in the Setup buffer of the response. For this request, it MUST be set to 0.

Reserved1 (1 byte): The client MUST set this value to 0.

Flags (2 bytes): An unused value that SHOULD be set to 0 when sending this message. The server MUST ignore this field when receiving this message.

Timeout (4 bytes): The maximum amount of time in milliseconds to wait for the operation to complete. The client MAY set this to 0 to indicate that no time-out is given. If the operation does not complete within the specified time, the server MAY abort the request and send a failure response.<128>

Reserved 2 (2 bytes): An unused value that SHOULD be set to 0 when sending this message. The server MUST ignore this field when receiving this message.

ParameterCount (2 bytes): This value MUST be set to the size (in bytes) of the parameter buffer. For this request, it MUST be set to 6 + length of the **FileName** field.

ParameterOffset (2 bytes): An offset in bytes from the start of the SMB header for this request to the start of the Parameter buffer. The Parameter buffer MUST be aligned to start on a 4-byte boundary from the start of the SMB header. The **ParameterOffset** MUST point to the offset of the **InformationLevel** field from the start of the SMB header.

DataCount (2 bytes): A count of bytes in the Data buffer field. For this request, it MUST be 0.

DataOffset (2 bytes): An offset in bytes to the start of the Data buffer field from the start of the SMB header for this request. This value MUST be greater than or equal to **ParameterOffset** plus **ParameterCount**. The Data buffer MUST be aligned to start on a 4-byte boundary from the start of the SMB header. If the **DataCount** field is 0, the client MAY set the **DataOffset** field to 0, or the client MAY set this field to a location in the packet after where the Parameter bytes would be placed, if present.<129>

SetupCount (1 byte): The number of setup words. For this request, it MUST be set to 1.

Reserved3 (1 byte): An unused value that SHOULD be set to 0 when sending this message. The server MUST ignore this field when receiving this message.

Subcommand (2 bytes): First setup word for this request. This field MUST be set to TRANS2_QUERY_PATH_INFORMATION, the subcommand to request. For a list of subcommands and values, see section 2.2.13.

ByteCount (2 bytes): This value MUST be the number of bytes that follows the **ByteCount** field in this packet.

Pad1 (3 bytes): Must be set to 3 bytes of padding to set the start of the Parameter buffer on a 4-byte boundary from the start of the SMB header. The client SHOULD initialize the padding with zeros. The server MUST ignore the padding.

InformationLevel (2 bytes): A number used to identify the file or directory related information being requested by a client. Corresponding to each information level, the server returns a specific structure to the client.

If the server supports the CAP_INFOLEVEL_PASSTHRU (the CAP_INFOLEVEL_PASSTHRU bit was set in the **Capabilities** field in the SMB_COM_NEGOTIATE response), the client MAY add the constant value SMB_INFO_PASSTHROUGH (as specified in section 2.2.13) to this field to pass a native Windows NT **InformationLevel** query directly to the server. Native Windows NT information levels are specified in [MS-FSCC] section 2.4.

In addition to the **InformationLevel** values specified in [CIFS] section 4.2.14, the extension provides new **InformationLevel** values that can be used to query information from the server.

Name	Value
SMB_QUERY_FILE_ALLOCATION_INFO	0x105
SMB_QUERY_FILE_END_OF_FILEINFO	0x106

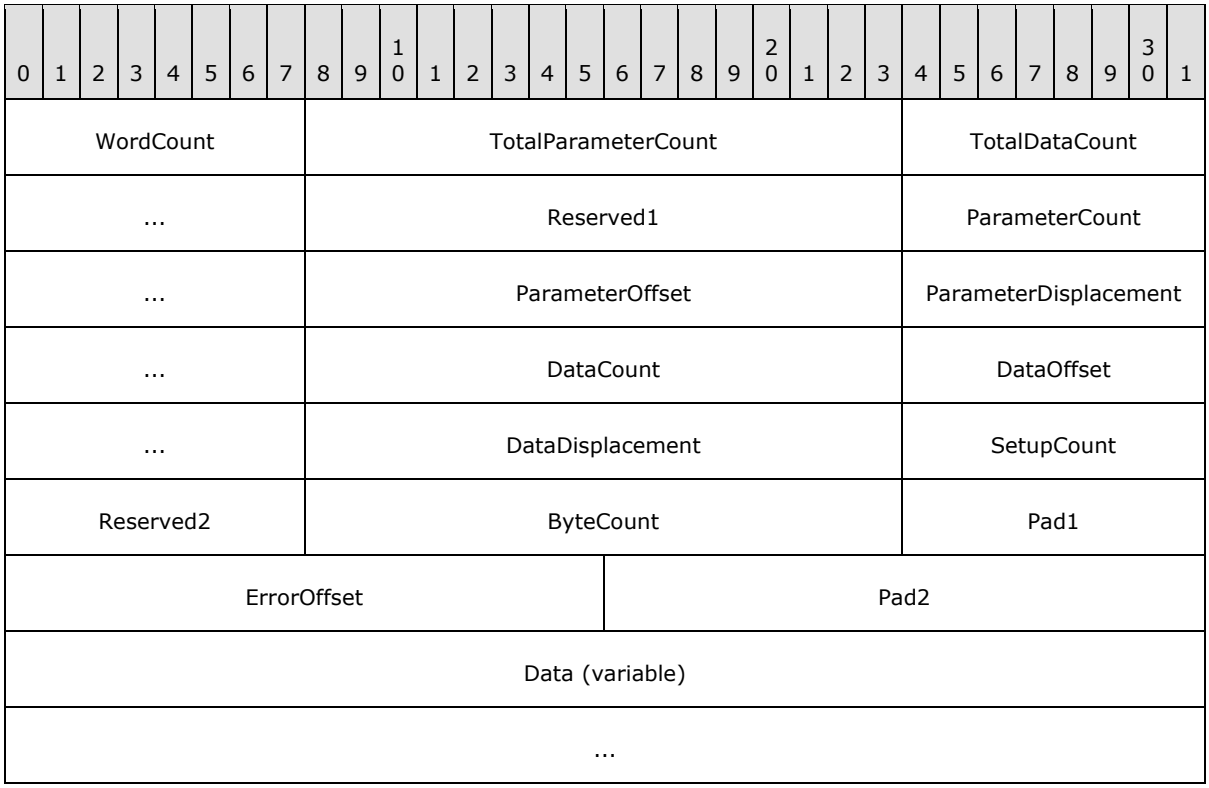
Reserved (4 bytes): An unused value that SHOULD be set to 0 when sending this request. The server MUST ignore this field when receiving this message.

FileName (variable): A string containing the fully qualified path. If the server supports UNICODE (the CAP_UNICODE field in the SMB_COM_NEGOTIATE response), the SMB clients MUST encode the string as a null-terminated array of 16-bit Unicode characters. Otherwise, it MUST encode the string as a null-terminated array of OEM characters.

2.2.13.4 TRANS2_QUERY_PATH_INFORMATION Response

A server MUST send a TRANS2_QUERY_FILE_INFORMATION response in reply to a SMB_COM_TRANSACTION2 client request with a TRANS2_QUERY_PATH_INFORMATION subcommand when the request is successful. The Data block of the transaction response contains the information requested, as specified in [CIFS] section 4.2.14.

If the **InformationLevel** field in the client request was SMB_INFO_PASSTHROUGH (as specified in section 2.2.13) or higher, the Data block is formatted with a native Windows NT-specific format. For more information, see section 3.2.4.7 and [MS-FSCC] section 2.4.



WordCount (1 byte): A count of 16-bit words in the response structure. For this response, this MUST be 0x0A (10), which is 10 plus the SetupCount of 0.

TotalParameterCount (2 bytes): The server MUST set this to 2.

TotalDataCount (2 bytes): The server MUST set this value to the number of bytes in the data buffer. The size of the buffer depends on the Windows NT-specific structure associated with the requested **InformationLevel**.

Reserved1 (2 bytes): An unused value that SHOULD be set to 0 when sending this response. The client MUST ignore this field when receiving this message.

ParameterCount (2 bytes): The server MUST set this value to 2.

ParameterOffset (2 bytes): An offset in bytes from the start of the SMB header for this response to the start of the **ErrorOffset** field contained in this response. The Parameter buffer MUST start after the **Pad1** field. The Parameter buffer MUST be aligned to start on a 4-byte boundary from the start of the SMB header.

ParameterDisplacement (2 bytes): An offset in bytes into the final Parameter buffer assembled from all responses. For a single buffer transaction (whose Parameter buffer fits in a single response), this value MUST be set to 0.

DataCount (2 bytes): The server MUST set this value to the number of bytes in the Data buffer contained in this packet. The size of this buffer is dependent on the **InformationLevel** in the corresponding TRANS2_QUERY_PATH_INFORMATION request.

DataOffset (2 bytes): An offset in bytes from the start of the SMB header for this response to the start of the Data buffer field. The Data buffer MUST be aligned to start on a 4-byte boundary from the start of the SMB header.

DataDisplacement (2 bytes): An offset in bytes into the final data buffer assembled from all responses. For a single buffer transaction (whose data buffer fits in a single response), this value MUST be set to 0.

SetupCount (1 byte): The number of setup words. For this response, it MUST be set to 0.

Reserved2 (1 byte): An unused value that SHOULD be set to 0 when sending this response. The client MUST ignore this field when receiving this message.

ByteCount (2 bytes): This value MUST be the number of bytes that follows the **ByteCount** field.

Pad1 (1 byte): Server MUST fill this 1-byte padding with zeros.

ErrorOffset (2 bytes): The server MUST set this field to 0.

Pad2 (2 bytes): The server MUST set this value to 0.

Data (variable): The Data buffer that contains the information bytes requested.

2.2.13.5 TRANS2_SET_FILE_INFORMATION Request

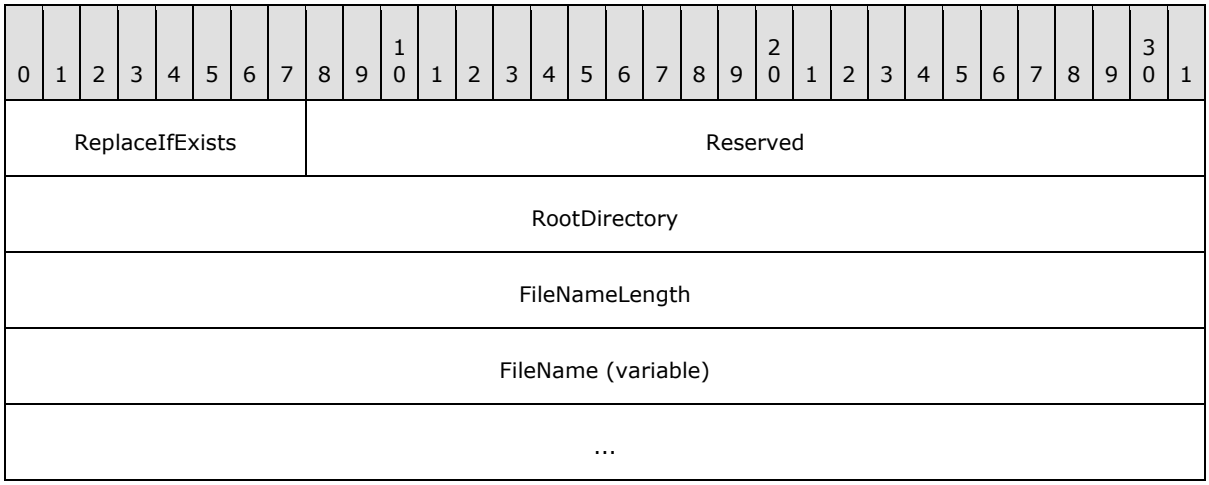
A TRANS2_SET_FILE_INFORMATION subcommand of the SMB_COM_TRANSACTION2 is sent by the client to request a change of attribute information for a file or directory that has been opened, as specified in [CIFS] section 4.2.17.

This extension provides a new **InformationLevel** value range that can be used to set information on the server. InformationLevel values that are SMB_INFO_PASSTHROUGH (for more information, see section 2.2.13) and higher indicate that the client is requesting the use of a native Windows NT

information level. For a set operation, the structure being applied is passed in the Data block of the transaction request, as specified in [CIFS] section 4.2.17. The format of the Data block is a native Windows NT-specific format that is dependent on the **InformationLevel**. Data format is as specified in [MS-FSCC] section 2.4 for all information levels except FileLinkInformation and FileRenameInformation, which are defined below in FILE_LINK_INFORMATION (section 2.2.13.5.1) and FILE_RENAME_INFORMATION (section 2.2.13.5.2) respectively.

2.2.13.5.1 FILE_LINK_INFORMATION

The FILE_LINK_INFORMATION information class is used to create an **NTFS** hard link to an existing file.



ReplaceIfExists (1 byte): An 8-bit field that is set to 1 to indicate that, if the link already exists, it SHOULD be replaced with the new link. MUST be set to 0 if the caller wants the link creation operation to fail if the link already exists.

Value	Meaning
0x00	If the link already exists, the link creation operation SHOULD fail.
0x01	If the link already exists, it SHOULD be replaced with the new link.

Reserved (3 bytes): Reserved for alignment.

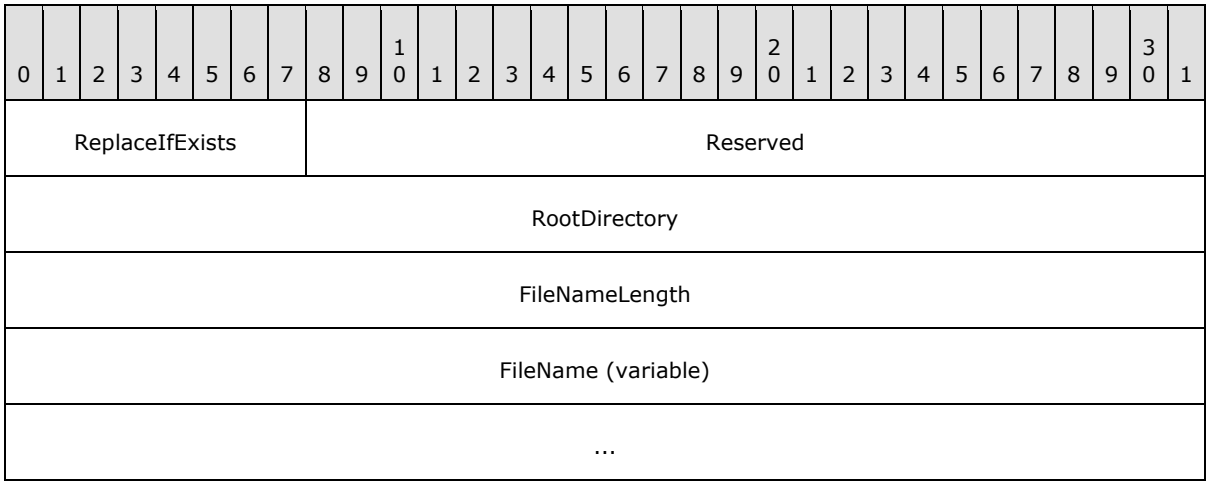
RootDirectory (4 bytes): A 32-bit unsigned integer that contains the file handle for the directory where the link is to be created. For network operations, this value MUST be zero.

FileNameLength (4 bytes): A 32-bit unsigned integer that contains the length, in bytes, of the **FileName** field.

FileName (variable): A sequence of **UNICODE** characters containing the name to be assigned to the newly-created link. This field might not be NULL-terminated and MUST be handled as a sequence of **FileNameLength** bytes. If the **RootDirectory** field is NULL and the link is to be created in a different directory from the file that is being linked to, this field specifies the full path name for the link to be created; otherwise, it specifies only the file name.

2.2.13.5.2 FILE_RENAME_INFORMATION

The FILE_RENAME_INFORMATION information class is used to rename a file.



ReplaceIfExists (1 byte): An 8-bit field that is set to 1 to indicate that, if a file with the given name already exists, it SHOULD be replaced with the given file. If set to 0, the rename operation MUST fail if a file with the given name already exists.

Value	Meaning
0x00	If a file with the given name already exists, the rename operation MUST fail.
0x01	If a file with the given name already exists, it SHOULD be replaced with the given file.

Reserved (3 bytes): Reserved area for alignment.

RootDirectory (4 bytes): A 32-bit unsigned integer that contains the file handle for the root directory. For network operations, this value MUST be zero.

FileNameLength (4 bytes): A 32-bit unsigned integer that contains the length, in bytes, of the new name for the file, including the trailing NULL if present.

FileName (variable): A sequence of **UNICODE** characters containing the file name. This field MAY NOT be NULL-terminated, and MUST be handled as a sequence of **FileNameLength** bytes, not as a NULL-terminated string. If the **RootDirectory** field is NULL and the file is being moved to a different directory, this member MUST specify the full path name to be assigned to the file. Otherwise, it MUST specify only the file name or a relative path name.

2.2.13.6 TRANS2_SET_FILE_INFORMATION Response

A server MUST send a TRANS2_SET_FILE_INFORMATION response in reply to an SMB_COM_TRANSACTION2 (section 2.2.13) client request with a TRANS2_SET_FILE_INFORMATION subcommand when the request is successful. The format for the SMB_COM_TRANSACTION2 response is as specified in [CIFS] section 3.13.1. There are no changes to the TRANS2_SET_FILE_INFORMATION response using the new **InformationLevel** values.

2.2.13.7 TRANS2_SET_PATH_INFORMATION Request

A TRANS2_SET_PATH_INFORMATION subcommand of the SMB_COM_TRANSACTION2 is sent by the client to request a change of attribute information for a file or directory by path, as specified in [CIFS] section 4.2.16.

This extension provides a new **InformationLevel** value range that can be used to set information on the server. **InformationLevel** values that are SMB_INFO_PASSTHROUGH (as specified in section 2.2.13) and higher indicate that the client is requesting the use of a native Windows NT information level. For a set operation, the structure being applied is passed in the Data block of the transaction request, as specified in [CIFS] section 4.2.16. The format of the Data block is a native Windows NT-specific format that is dependent on the **InformationLevel**. For more information, see [MS-FSCC] section 2.4.

2.2.13.8 TRANS2_SET_PATH_INFORMATION Response

A server MUST send a TRANS2_SET_PATH_INFORMATION response in reply to an SMB_COM_TRANSACTION2 (section 2.2.13) client request with a TRANS2_SET_PATH_INFORMATION subcommand when the request is successful. The format for the SMB_COM_TRANSACTION2 response is as specified in [CIFS] section 3.13.1. There are no changes to the TRANS2_SET_PATH_INFORMATION response using the new **InformationLevel** values.

2.2.13.9 TRANS2_QUERY_FS_INFORMATION Request

A TRANS2_QUERY_FS_INFORMATION subcommand of the SMB_COM_TRANSACTION2 is sent by the client to request attribute information about the file system, as specified in [CIFS] section 4.1.6.

This extension provides a new **InformationLevel** value range that can be used to query information from the server. **InformationLevel** values that are SMB_INFO_PASSTHROUGH (as specified in section 2.2.13) and higher indicate that the client is requesting a native Windows NT information level.

2.2.13.10 TRANS2_QUERY_FS_INFORMATION Response

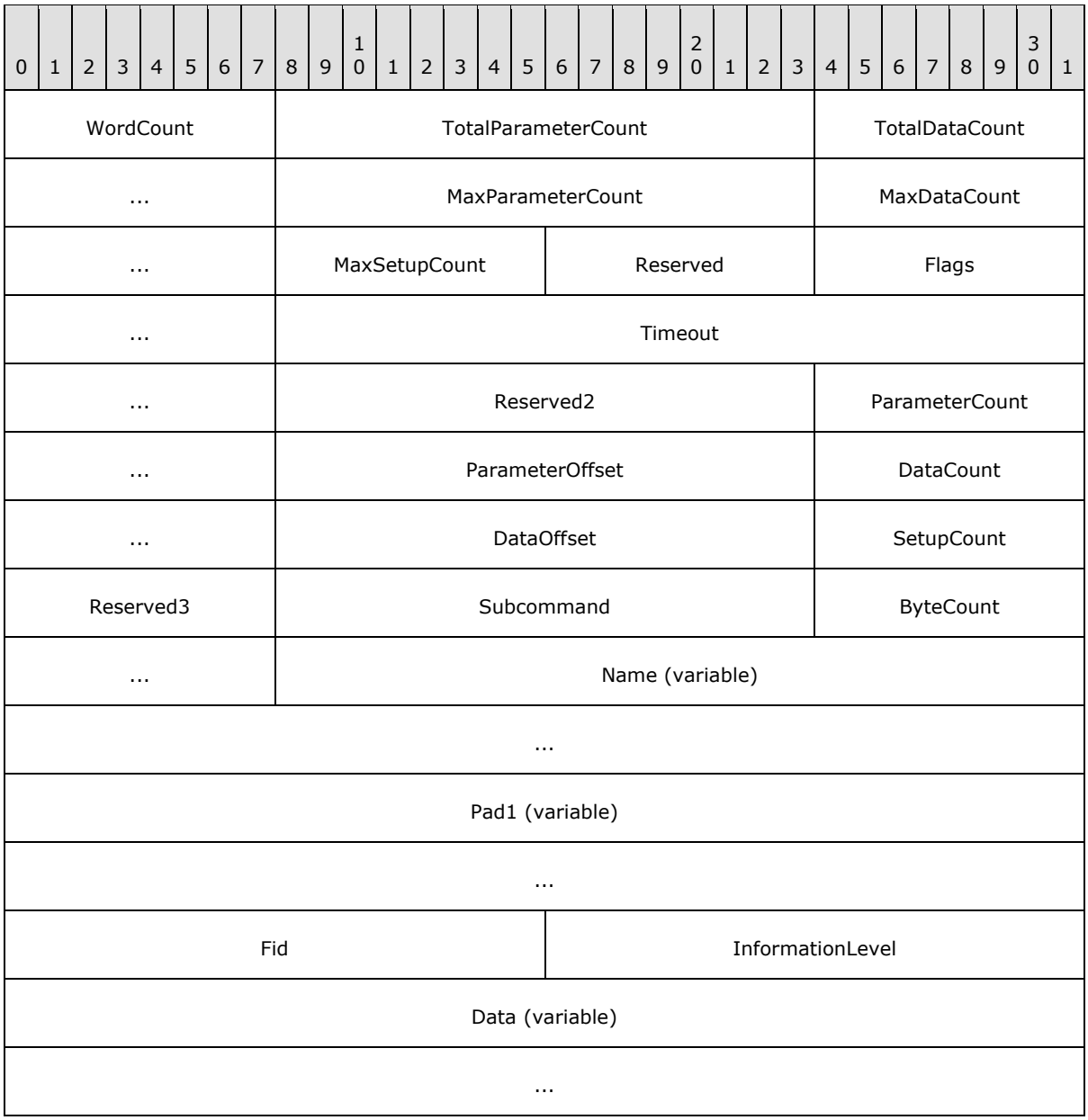
A server MUST send a TRANS2_QUERY_FS_INFORMATION response in reply to an SMB_COM_TRANSACTION2 (section 2.2.13) client request with a TRANS2_QUERY_FS_INFORMATION subcommand when the request is successful. The Data block of the transaction response contains the information requested, as specified in [CIFS] section 4.1.6.

If the **InformationLevel** field in the client request is SMB_INFO_PASSTHROUGH (as specified in section 2.2.13) and higher, the Data block is formatted with a native Windows NT-specific format. For more information, see section 3.2.4.7 and [MS-FSCC] section 2.5.

2.2.13.11 TRANS2_SET_FS_INFORMATION Request

A TRANS2_SET_FS_INFORMATION subcommand of the SMB_COM_TRANSACTION2 (section 2.2.13) is sent by the client to request a change of file system attributes. This request is new to these extensions and similar in nature to the traditional TRANS2_SET_FILE_INFORMATION request (as specified in [CIFS] section 4.2.17) in format.

The format follows the convention for SMB_COM_TRANSACTION2, as specified in [CIFS] section 3.13.1.



WordCount (1 byte): A count of 16-bit words in the request structure. For this request, it MUST be 0x0F (15), which is 14 plus the **SetupCount** of 1.

TotalParameterCount (2 bytes): The number of bytes in the Parameter buffer. For this request, it MUST be 4. The content of the Parameter buffer is described below.

TotalDataCount (2 bytes): The number of bytes in the Data buffer. For this request, it MUST be set to 0.

MaxParameterCount (2 bytes): The maximum number of bytes the client will accept in the Parameter buffer of the response. For this request, it MUST be set to 0.

MaxDataCount (2 bytes): The maximum number of bytes the client will accept in the Data buffer of the response. For this request, it MUST be set to 0.

MaxSetupCount (1 byte): The maximum number of bytes the client will accept in the Setup buffer of the response. For this request, it MUST be set to 0.

Reserved (1 byte): An unused value that SHOULD be set to 0 when sending this message. The server MUST ignore this field when receiving this message.

Flags (2 bytes): A set of options that provide additional information to the server on this request. The client MAY set either of the following bits. Unused bit fields SHOULD be set to 0 by the client when sending a request, and MUST be ignored when received by the server.<130>

Value	Meaning
DISCONNECT_TID 0x0001	The server MUST disconnect the tree connect associated with the Tid received in the SMB header of this request after the request completes. The client SHOULD NOT send an SMB_COM_TREE_DISCONNECT for this tree connect.
NO_RESPONSE 0x0002	The server MUST process this client request as a one-way transaction and MUST NOT send a response back to the client.<131>

Timeout (4 bytes): The maximum amount of time in milliseconds to wait for the operation to complete. The client MAY set this to 0 to indicate that no time-out is given. If the operation does not complete within the specified time, the server MAY abort the request and send a failure response.<132>

Reserved2 (2 bytes): An unused value that SHOULD be set to 0 when sending this message. The server MUST ignore this field when receiving this message.

ParameterCount (2 bytes): A count of bytes in the Parameter buffer of this packet. For this request, it MUST be less than or equal to 4.

ParameterOffset (2 bytes): An offset in bytes from the start of the SMB header for this request to the start of the Parameter buffer. The Parameter buffer MUST start after the **Subcommand** field. The Parameter buffer MUST be aligned to start on a 4-byte boundary from the start of the SMB header.

DataCount (2 bytes): The number of bytes in the Data buffer field.

DataOffset (2 bytes): An offset in bytes to the start of Data buffer field from the start of the SMB header for this request. This value MUST be greater than or equal to **ParameterOffset** plus **ParameterCount**. The Data buffer MUST be aligned to start on a 4-byte boundary from the start of the SMB header.

SetupCount (1 byte): The number of setup words. For this request, it MUST be set to 1.

Reserved3 (1 byte): An unused value that SHOULD be set to 0 when sending this message. The server MUST ignore this field when receiving this message.

Subcommand (2 bytes): The first setup word for this request. This field MUST be set to TRANS2_SET_FS_INFORMATION, the subcommand to request. For a list of subcommands and values, see section 2.2.13.

ByteCount (2 bytes): This value MUST be set to the number of bytes that follows the **ByteCount** field in this packet.

Name (variable): The name for this transaction that MUST be NULL for SMB_COM_TRANSACTION2 requests. If SMB_FLAGS2_UNICODE is set in the **Flags2** field of the SMB header of the request, the name string MUST be a null-terminated array of 16-bit Unicode characters. Otherwise, the name string MUST be a null-terminated array of OEM characters. The server MUST ignore this field in the request. If the name string consists of Unicode characters, this field MUST be aligned to start on a 2-byte boundary from the start of the SMB header.

Pad1 (variable): Optional padding to set the Parameter buffer to start on a 4-byte boundary from the start of the SMB header.

Fid (2 bytes): First parameter in the Parameter buffer for this request. This field is the SMB file identifier for the file system that is having its file attributes changed. This field MUST be set to a valid Fid from a server response for a previous SMB command to open or create a file. These commands include SMB_COM_OPEN, SMB_COM_CREATE, SMB_COM_CREATE_TEMPORARY, SMB_COM_CREATE_NEW, SMB_COM_OPEN_ANDX, SMB_COM_NT_CREATE_ANDX, and SMB_COM_NT_TRANSACT with subcommand NT_TRANSACT_CREATE.

InformationLevel (2 bytes): The second parameter in the Parameter buffer for this request. This field is the level of information to be applied. The **InformationLevel** field passed in this request MUST be set to a 16-bit value greater than or equal to SMB_INFO_PASSTHROUGH (as specified in section 2.2.13). The information set using this range of **InformationLevel** is native Windows NT-specific information, as specified in [MS-FSCC] section 2.5.

Data (variable): The value of the file system attributes being applied MUST be passed in the Data buffer of the transaction request. The format of this structure is determined by the value of the **InformationLevel** field.

2.2.13.12 TRANS2_SET_FS_INFORMATION Response

A server MUST send a TRANS2_SET_FS_INFORMATION response in reply to a client TRANS2_SET_FS_INFORMATION (section 2.2.13.11) subcommand request when the request is successful. The server MUST set an error code in the Status field of SMB header of the response to indicate whether the request was successful or failed.

The response format follows the convention for a Server Response, as specified in [CIFS] section 3.13.1.

0	1	2	3	4	5	6	7	8	9	10	1	2	3	4	5	6	7	8	9	20	1	2	3	4	5	6	7	8	9	30	1
WordCount								TotalParameterCount																TotalDataCount							
...								Reserved																ParameterCount							
...								ParameterOffset																ParameterDisplacement							
...								DataCount																DataOffset							
...								DataDisplacement																SetupCount							
Reserved2								ByteCount																Pad							

WordCount (1 byte): A count of 16-bit words in the response structure. For this response, this MUST be 0x0A (10), which is 10 plus the SetupCount of zero.

TotalParameterCount (2 bytes): The number of bytes in the Parameter buffer. For this response, it MUST be set to 0.

TotalDataCount (2 bytes): The number of bytes in the Data buffer. For this response, it MUST be set to 0.

Reserved (2 bytes): An unused value that SHOULD be set to 0 when sending this response. The client MUST ignore this field when receiving this message.

ParameterCount (2 bytes): The number of bytes of the Parameter buffer contained in this packet. For this response, it MUST be set to 0.

ParameterOffset (2 bytes): An offset in bytes from the start of the SMB header for this response to the start of the part of the Parameter buffer contained in this response. The Parameter buffer MUST start after the Unused2 field. The Parameter buffer MUST be aligned to start on a 4-byte boundary from the start of the SMB header. If the ParameterCount field is 0, the server MAY set the ParameterOffset field to 0, or the server MAY set this field to a location in the packet after where Parameter bytes would have been placed, if present.<133>

ParameterDisplacement (2 bytes): An offset in bytes into the final Parameter buffer assembled from all responses. For a single buffer transaction (whose Parameter buffer fits in a single response), this value MUST be set to 0.

DataCount (2 bytes): The number of bytes in the Data buffer contained in this packet. For this response, it MUST be set to 0.

DataOffset (2 bytes): An offset in bytes from the start of the SMB header for this response to the start of the Data buffer field. This value MUST be greater than or equal to ParameterOffset plus ParameterCount. The Data buffer MUST be aligned to start on a 4-byte boundary from the start of the SMB header. If the **DataCount** field is 0, the server MAY set the DataOffset field to 0, or the server MAY set this field to a location in the packet after where the Parameter bytes would be placed, if present.<134>

DataDisplacement (2 bytes): An offset in bytes into the final Data buffer assembled from all responses. For a single buffer transaction (whose Data buffer fits in a single response), this value MUST be set to 0.

SetupCount (1 byte): The number of setup words. For this response, it MUST be set to 0.

Reserved2 (1 byte): An unused value that SHOULD be set to 0 when sending this response. The client MUST ignore this field when receiving this message.

ByteCount (2 bytes): This value MUST be the number of bytes that follows the ByteCount field.

Pad (1 byte): Server MUST set this to zero.

2.2.13.13 TRANS2_FIND_FIRST2 Request

A TRANS2_FIND_FIRST2 (section 2.2.13.13) subcommand of the SMB_COM_TRANSACTION2 (section 2.2.13) request is sent by the client to retrieve an enumeration of files, as specified in [CIFS] section 4.3.4. The request format is identical to the format that is specified in [CIFS], with the exception that there are two new **InformationLevel** values, SMB_FIND_FILE_ID_FULL_DIRECTORY_INFO (section 2.2.13.14.1) and SMB_FIND_FILE_ID_BOTH_DIRECTORY_INFO (section 2.2.13.14.2), that can be requested. All levels are included in the following table for completeness. For details about the other levels, see [CIFS] sections 4.3.4.1 through 4.3.4.7.

The **InformationLevel** field MUST be set to one of the following values.

InformationLevel	Value
SMB_INFO_STANDARD	0x0001
SMB_INFO_QUERY_EA_SIZE	0x0002
SMB_INFO_QUERY_EAS_FROM_LIST	0x0003
SMB_FIND_FILE_DIRECTORY_INFO	0x0101
SMB_FIND_FILE_FULL_DIRECTORY_INFO	0x0102
SMB_FIND_FILE_NAMES_INFO	0x0103
SMB_FIND_FILE_BOTH_DIRECTORY_INFO	0x0104
SMB_FIND_FILE_ID_FULL_DIRECTORY_INFO	0x0105
SMB_FIND_FILE_ID_BOTH_DIRECTORY_INFO	0x0106

2.2.13.14 TRANS2_FIND_FIRST2 Response

A server MUST send a TRANS2_FIND_FIRST2 response in reply to a client TRANS2_FIND_FIRST2 subcommand request when the request is successful. This response is identical to the format specified in [CIFS] section 4.3.4, with the exception of the two new **InformationLevels**, as defined in section 2.2.13.13. The format of the file information returned for these two information levels (and the format of information for a FIND_FIRST2 for the special FileName pattern @GMT-*) are listed in the sections that follow this section.

2.2.13.14.1 SMB_FIND_FILE_ID_FULL_DIRECTORY_INFO

The fields and encoding of the TRANS2_FIND_FIRST2 (section 2.2.13.14) response message are identical to the fields and encoding of SMB_FIND_FILE_FULL_DIRECTORY_INFORMATION, as specified in [CIFS] section 4.3.4.5, with the addition of the **FileId** field described in the list that follows the table.

0	1	2	3	4	5	6	7	8	9	10	1	2	3	4	5	6	7	8	9	20	1	2	3	4	5	6	7	8	9	30	1
NextEntryOffset																															
FileIndex																															
CreationTime																															
...																															
LastAccessTime																															
...																															
LastWriteTime																															
...																															
ChangeTime																															
...																															
EndOfFile																															
...																															
AllocationSize																															
...																															
ExtFileAttributes																															
FileNameLength																															
EaSize																															

FileId
...
FileName (variable)
...

NextEntryOffset (4 bytes): The offset in bytes from the beginning of this element to the beginning of the next element, or 0 if this is the final element.

FileIndex (4 bytes): This field MUST be set to 0 when sending a response, and MUST be ignored when the client receives this message.

CreationTime (8 bytes): A **LARGE_INTEGER** value that contains a time stamp.

LastAccessTime (8 bytes): A **LARGE_INTEGER** value that contains a time stamp.

LastWriteTime (8 bytes): A **LARGE_INTEGER** value that contains a time stamp.

ChangeTime (8 bytes): A **LARGE_INTEGER** value that contains a time stamp.

EndOfFile (8 bytes): This **LARGE_INTEGER** field MUST be set to 0 when sending a response, and MUST be ignored when the client receives this message.

AllocationSize (8 bytes): This **LARGE_INTEGER** field MUST be set to 0 when sending a response, and MUST be ignored when the client receives this message.

ExtFileAttributes (4 bytes): Extended attributes for this file that MUST be marked as a DIRECTORY.

FileNameLength (4 bytes): The length, in bytes, of the **FileName** field.

EaSize (4 bytes): This field MUST be set to 0 when sending a response, and MUST be ignored when the client receives this message.

FileId (8 bytes): A **LARGE_INTEGER** that serves as an internal file system identifier. This number MUST be unique for each file on a given volume. If a remote file system does not support unique **FileId** values, the **FileId** field MUST be set to 0.

FileName (variable): The full name for the file. If SMB_FLAGS2_UNICODE is set in the **Flags2** field of the SMB header of the request, the **FileName** string MUST be an array of 16-bit Unicode characters and is not null terminated. Otherwise, the name string MUST be an array of OEM characters and is not null terminated. If the name string consists of Unicode characters, this field MUST be aligned to start on a 2-byte boundary from the start of the SMB header.

2.2.13.14.2 SMB_FIND_FILE_ID_BOTH_DIRECTORY_INFO

The fields and encoding of the TRANS2_FIND_FIRST2 (section 3.2.5.9) response message are identical to the fields and encoding of SMB_FIND_FILE_BOTH_DIRECTORY_INFORMATION, as specified in [CIFS] section 4.3.4.6, with the addition of the **FileId** field described in the list that follows the table.

...	
...	
...	
...	
...	FileId
...	
...	FileName (variable)
...	

NextEntryOffset (4 bytes): The offset in bytes from the beginning of this element to the beginning of the next element, or zero if this is the final element.

FileIndex (4 bytes): This field MUST be set to 0 when sending a response, and MUST be ignored when the client receives this message.

CreationTime (8 bytes): A **LARGE_INTEGER** value containing a time stamp.

LastAccessTime (8 bytes): A **LARGE_INTEGER** value containing a time stamp.

LastWriteTime (8 bytes): A **LARGE_INTEGER** value containing a time stamp.

ChangeTime (8 bytes): A **LARGE_INTEGER** value containing a time stamp.

EndOfFile (8 bytes): This **LARGE_INTEGER** field MUST be set to 0 when sending a response, and MUST be ignored when the client receives this message.

AllocationSize (8 bytes): This **LARGE_INTEGER** field MUST be set to 0 when sending a response, and MUST be ignored when the client receives this message.

ExtFileAttributes (4 bytes): Extended attributes for this file that MUST be marked as a DIRECTORY.

FileNameLength (4 bytes): Length, in bytes, of the **FileName** field.

EaSize (4 bytes): This field MUST be set to 0 when sending a response, and MUST be ignored when the client receives this message.

ShortNameLength (1 byte): Length, in bytes, of ShortName field.

ShortName (24 bytes): 8.3 name for the file. The ShortName is field formatted as an array of 16-bit Unicode characters and is not null terminated.

FileId (8 bytes): A **LARGE_INTEGER** that serves as an internal file system identifier. This number MUST be unique for each file on a given volume. If a remote file system does not support unique FileId values, the FileId field MUST be set to 0.

FileName (variable): The full name for the file. If SMB_FLAGS2_UNICODE is set in the Flags2 field of the SMB header of the request, the FileName string MUST be an array of 16-bit Unicode characters and is not null terminated. Otherwise, the name string MUST be an array of OEM characters and is not null terminated. If the name string consists of Unicode characters, this field MUST be aligned to start on a two-byte boundary from the start of the SMB header.

2.2.13.14.3 SMB_FIND_FILE_BOTH_DIRECTORY_INFO for Previous File Version Enumeration

A client implementing optional support for previous versions via TRANS2_FIND_FIRST2 and TRANS2_FIND_NEXT2 (for more information, see section 3.2.4.6) MAY issue a request for SMB_FIND_FILE_BOTH_DIRECTORY_INFO using the special FileName pattern @GMT-* passed in the **FileName** field in the parameter block (as specified in section 4.2) to indicate a request to enumerate previous version snapshots.<135>

If the query being executed is a previous version enumeration, as defined above, the returned structure is identical to the structure that is specified in [CIFS] section 4.3.4.6. However, the fields have a slightly different definition as defined in the following.

0	1	2	3	4	5	6	7	8	9	10	1	2	3	4	5	6	7	8	9	20	1	2	3	4	5	6	7	8	9	30	1
NextEntryOffset																															
FileIndex																															
CreationTime																															
...																															
LastAccessTime																															
...																															
LastWriteTime																															
...																															
ChangeTime																															
...																															

EndOfFile	
...	
AllocationSize	
...	
ExtFileAttributes	
FileNameLength	
EaSize	
ShortNameLength	ShortName
...	
...	
...	
...	
...	
...	FileName (variable)
...	

NextEntryOffset (4 bytes): The offset in bytes from the beginning of this element to the beginning of the next element, or zero if this is the final element.

FileIndex (4 bytes): This field MUST be set to 0 when sending a response, and MUST be ignored when the client receives this message.

CreationTime (8 bytes): Time stamp represented by the @GMT token.

LastAccessTime (8 bytes): A **LARGE_INTEGER** time stamp represented by the @GMT token.

LastWriteTime (8 bytes): A **LARGE_INTEGER** time stamp represented by the @GMT token.

ChangeTime (8 bytes): A **LARGE_INTEGER** time stamp represented by the @GMT token.

EndOfFile (8 bytes): This field MUST be set to 0 when sending a response, and MUST be ignored when the client receives this message.

AllocationSize (8 bytes): This **LARGE_INTEGER** field MUST be set to 0 when sending a response, and MUST be ignored when the client receives this message.

ExtFileAttributes (4 bytes): Extended attributes for this file that MUST be marked as a DIRECTORY.

FileNameLength (4 bytes): Length, in bytes, of the **FileName** field.

EaSize (4 bytes): This field MUST be set to 0 when sending a response, and MUST be ignored when the client receives this message.

ShortNameLength (1 byte): Length, in bytes, of **ShortName** field.

ShortName (24 bytes): 8.3 name for the file formatted as @GMT~XXX where XXX is an array index value in decimal of an array of snapshots returned, starting at an array index value of zero. The **ShortName** is field formatted as an array of 16-bit Unicode characters and is not null terminated.

FileName (variable): Full name for the file formatted as an @GMT token. If SMB_FLAGS2_UNICODE is set in the Flags2 field of the SMB header of the request, the FileName string MUST be an array of 16-bit Unicode characters and is not null terminated. Otherwise, the name string MUST be an array of OEM characters and is not null terminated. If the name string consists of Unicode characters, this field MUST be aligned to start on a two-byte boundary from the start of the SMB header.

2.2.13.15 TRANS2_FIND_NEXT2 Request

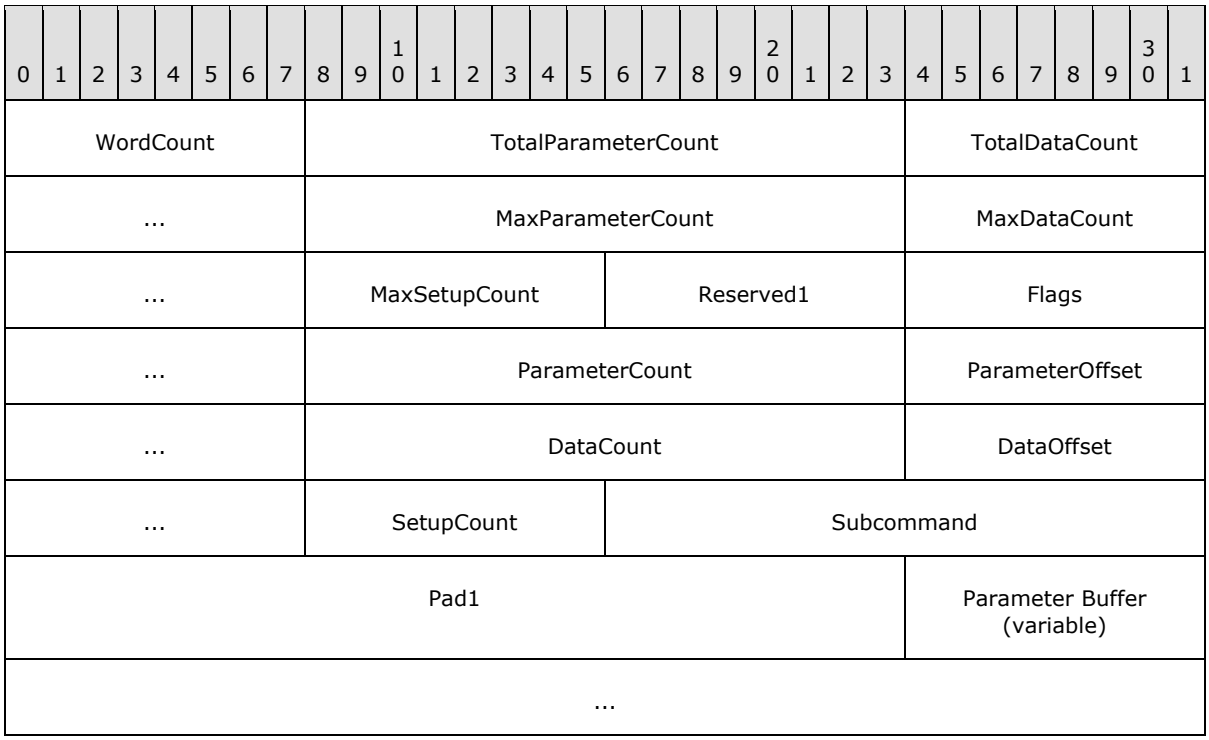
The TRANS2_FIND_NEXT2 subcommand of the SMB_COM_TRANSACTION2 request is sent by a client to continue a file enumeration, as specified in [CIFS] section 4.3.5. The request format is identical to the request format that is specified in [CIFS], with the exception that there are two new **InformationLevel** values, SMB_FIND_FILE_ID_FULL_DIRECTORY_INFO and SMB_FIND_FILE_ID_BOTH_DIRECTORY_INFO, that can be requested, as specified in section 2.2.13.13.

2.2.13.16 TRANS2_FIND_NEXT2 Response

The server MUST send a TRANS2_FIND_NEXT2 response in reply to a client TRANS2_FIND_NEXT2 subcommand request when the request is successful. The format of this packet is identical to what is specified in [CIFS] section 4.3.5, except for the use of two new information levels. The format of the data block for these levels is as specified in section 2.2.13.14.

2.2.13.17 TRANS2_GET_DFS_REFERRAL Request

A TRANS2_GET_DFS_REFERRAL (section 2.2.13.17) subcommand of a SMB_COM_TRANSACTION2 (section 2.2.13) request is sent by a client in response to the higher level event described in section 3.2.4.12. The application supplies the payload for the request. The payload MUST be formatted as described in [MS-DFSC]. The format follows the convention for SMB_COM_TRANSACTION2 (section 2.2.13), as specified in [CIFS] section 3.13.1.



WordCount (1 byte): A count of 16-bit words in the request structure. For this request, the count MUST be 0x15 (21), which is 19 plus the **SetupCount** value of 2.

TotalParameterCount (2 bytes): The number of bytes in the Parameter buffer. The content of the Parameter buffer is described below.

TotalDataCount (2 bytes): The number of bytes in the Data buffer. For this request, the count MUST be set to 0.

MaxParameterCount (2 bytes): The maximum number of bytes the client will accept in the Parameter buffer of the response. For this request, the count MUST be set to 0.

MaxDataCount (2 bytes): The maximum number of bytes the client will accept in the Data buffer of the response. For this request, it MUST be set to the size in bytes of the application provided buffer into which to copy the response.

MaxSetupCount (1 byte): The maximum number of bytes the client will accept in the Setup buffer of the response. For this request, the count MUST be set to 0.

Reserved1 (1 byte): Client MUST set this value to 0.

Flags (2 bytes): An unused value that SHOULD be set to 0 when sending this message. The server MUST ignore this field when receiving this message.

ParameterCount (2 bytes): This MUST be set to the size (in bytes) of the parameter buffer. It should be the same as the TotalParameterCount if it is a single message transaction.

ParameterOffset (2 bytes): An offset in bytes from the start of the SMB header for this request to the start of the Parameter buffer. The Parameter buffer MUST be aligned to start on

a 4-byte boundary from the start of the SMB header. The ParameterOffset MUST point to the start of the application supplies payload.

DataCount (2 bytes): A count of bytes in the Data buffer field. For this request, it MUST be 0.

DataOffset (2 bytes): An offset in bytes to the start of the Data buffer field from the start of the SMB header for this request. This value MUST be greater than or equal to ParameterOffset plus ParameterCount. The Data buffer MUST be aligned to start on a 4-byte boundary from the start of the SMB header. If the DataCount field is 0, the client MAY set the DataOffset field to 0, or the client MAY set this field to a location in the packet after where the Parameter bytes would be placed, if present.<136>

SetupCount (1 byte): The number of setup words. For this request, the count MUST be set to 1.

Subcommand (2 bytes): First setup word for this request. This field MUST be set to TRANS2_GET_DFS_REFERRAL (section 2.2.13.17).

Pad1 (3 bytes): Must be set to three bytes of padding to set the start of the Parameter buffer on a 4-byte boundary from the start of the SMB header. The client SHOULD initialize the padding with zeros. The server MUST ignore the padding.

Parameter Buffer (variable): Contains application payload.

2.2.13.18 TRANS2_GET_DFS_REFERRAL Response

A server MUST send a TRANS2_GET_DFS_REFERRAL response in reply to a client TRANS2_GET_DFS_REFERRAL (section 2.2.13.17) subcommand request when the request is successful.

The format of a TRANSACT2 response is specified in [CIFS] section 3.13.1. The data buffer of the TRANSACT2 response MUST be formatted as specified in [MS-DFSC]. The SMB clients MUST copy the data buffer from the response into an application supplied buffer specified in section 3.2.4.12. The TRANSACT2 response MUST NOT have setup or parameter buffers.

2.2.14 SMB_COM_NT_TRANSACTION Extensions and Clarification

The SMB_COM_NT_TRANSACTION request is sent by a client to specify operations on the server, including file open, file create, device I/O control, file rename, notify directory change, and set and query security descriptors. The possible subcommand codes in the SMB_COM_NT_TRANSACTION client request are as specified in [CIFS] section 6.3.

The extensions to the CIFS Protocol add two new subcommand codes, NT_TRANSACT_QUERY_QUOTA (section 2.2.14.3) and NT_TRANSACT_SET_QUOTA, to SMB_COM_NT_TRANSACTION. In addition, the CIFS Protocol defines a subcommand (NT_TRANSACT_RENAME) but does not define the request format. The request format is defined in this document.

For completeness, the entire list of subcommands is included below. The client MUST send a subcommand listed below, and the server MUST fail a request for a subcommand not listed with STATUS_INVALID_PARAMETER.

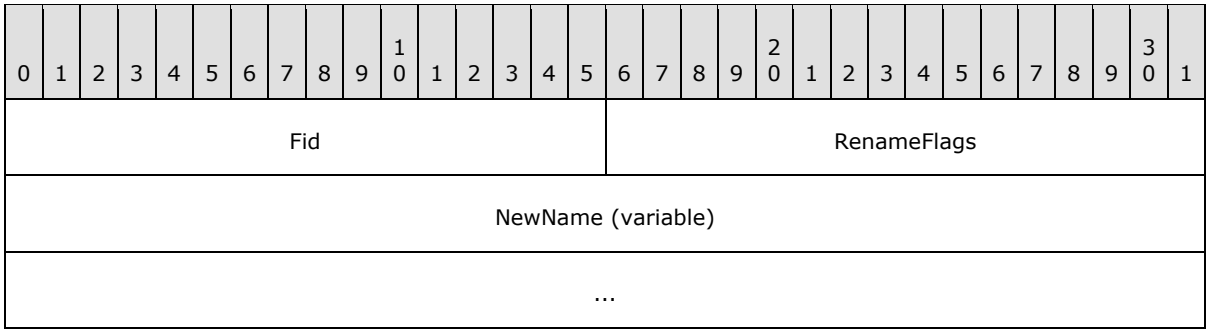
Subcommand code	Value	Description
NT_TRANSACT_CREATE	0x0001	As specified in [CIFS] section 4.2.2.

Subcommand code	Value	Description
NT_TRANSACT_IOCTL	0x0002	As specified in [CIFS] section 4.6.1.
NT_TRANSACT_SET_SECURITY_DESC	0x0003	As specified in [CIFS] section 4.6.3.
NT_TRANSACT_NOTIFY_CHANGE	0x0004	As specified in [CIFS] section 4.3.6.
NT_TRANSACT_RENAME	0x0005	Rename request, as specified in section 2.2.14.1.
NT_TRANSACT_QUERY_SECURITY_DESC	0x0006	As specified in [CIFS] section 4.6.2.
NT_TRANSACT_QUERY_QUOTA	0x0007	Query a server for quota information, as specified in section 2.2.14.3.
NT_TRANSACT_SET_QUOTA	0x0008	Set quota information on a server, as specified in section 2.2.14.5.

2.2.14.1 NT_TRANSACT_RENAME Client Request Clarification

An SMB_COM_NT_TRANSACTION (section 2.2.14) request with an NT_TRANSACT_RENAME subcommand code is sent by a client to request that a file or directory be renamed on the server. The NT_TRANSACT_RENAME subcommand code is listed in [CIFS] section 6.3, but the request format is not defined in [CIFS].

The **TotalParameterCount** MUST be greater than or equal to 5 with the Parameter block containing the following structure.



Fid (2 bytes): The SMB file identifier of the file to rename.

RenameFlags (2 bytes): The flags for the rename, defined below.

NewName (variable): The name that the file is being renamed to. If SMB_FLAGS2_UNICODE is set in the **Flags2** field of the SMB header of the request, the **NewName** field MUST be a null-terminated array of 16-bit Unicode characters. Otherwise, the **NewName** field MUST be a null-terminated array of OEM characters. If the **NewName** string consists of Unicode characters, this field MUST be aligned to start on a two-byte boundary from the start of the SMB header.

The **RenameFlags** field has only one possible flag value. If **RenameFlags** is set to 0x0001, the rename operation MUST replace the target file if the target name exists. If **RenameFlags** is set to 0x0000, the rename operation MUST fail if the target name exists. All other bit fields in the **RenameFlags** field SHOULD be set to 0 by the client when sending the request; and the server MUST ignore other bit fields when the message is received.<137>

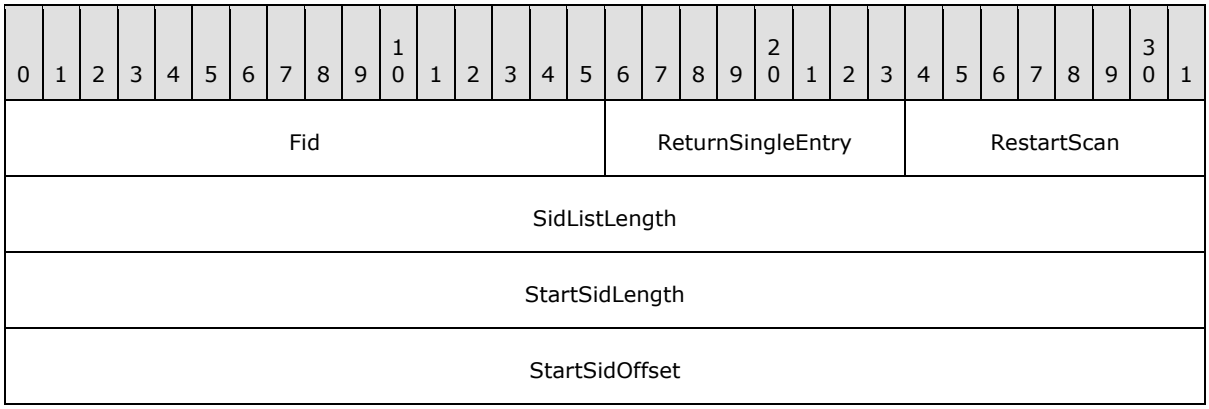
2.2.14.2 NT_TRANSACT_RENAME Server Response Clarification

A server MUST send an NT_TRANSACT_RENAME response in reply to a client NT_TRANSACT_RENAME (section 2.2.14.1) subcommand request when the request is successful. The server transaction response to the NT_TRANSACT_RENAME request MUST have the **TotalParameterCount** and **TotalDataCount** fields set to 0. Only the status code is used to determine success/failure of the operation.

2.2.14.3 NT_TRANSACT_QUERY_QUOTA Client Request Extension

An SMB_COM_NT_TRANSACTION (section 2.2.14) request with an NT_TRANSACT_QUERY_QUOTA subcommand code is sent by a client to query quota information from a server.

The **TotalParameterCount** field MUST be equal to 16, and the parameter block in the client request MUST be encoded with the following parameter block encoding fields.



Fid (2 bytes): The SMB file identifier of the target directory.

ReturnSingleEntry (1 byte): Indicates only a single entry is to be returned instead of filling the entire buffer.

RestartScan (1 byte): Indicates that the scan of the quota information is to be restarted.

SidListLength (4 bytes): Supplies the length in bytes of the SidList (see below), or 0 if there is no SidList.

StartSidLength (4 bytes): Supplies the length in bytes of the StartSid (see below), or 0 if there is no StartSid. MUST be ignored by the receiver if **SidListLength** is non-zero.

StartSidOffset (4 bytes): Supplies the offset, in bytes, to the StartSid in the Parameter buffer.

At least one of **SidListLength** and **StartSidLength** MUST be 0. If both are 0, this indicates that all SIDs are to be enumerated by the server as if they were passed in the SidList.

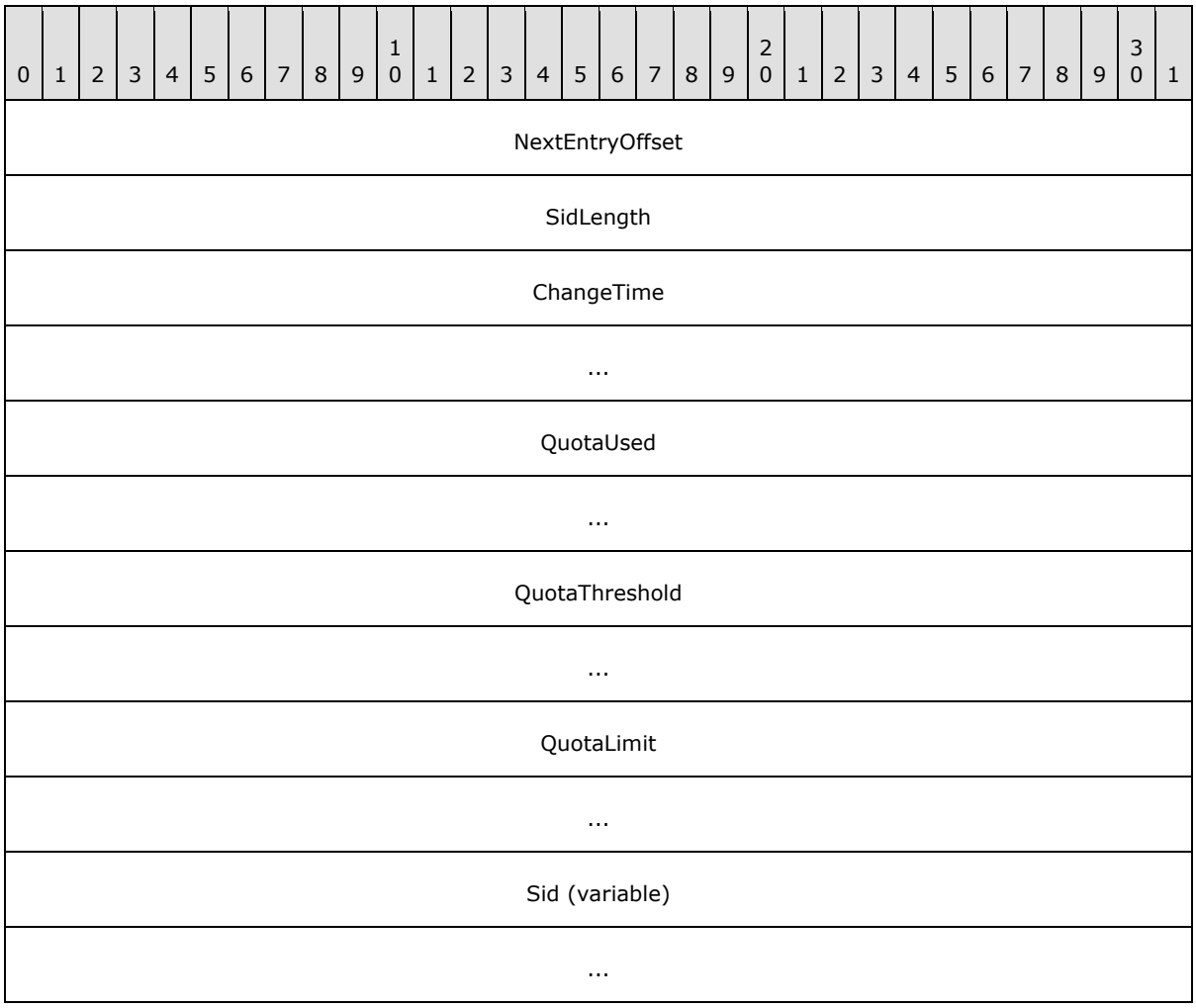
The **TotalDataCount** MUST indicate the size of the Data buffer with that buffer encoded as follows. If **SidListLength** is non-zero, the Data buffer contains a SidList, being a list of SIDs for which information is requested. If **StartSidLength** is non-zero, the Data buffer instead contains the StartSid (that is, the SID at which an enumeration is requested to start).

2.2.14.4 NT_TRANSACT_QUERY_QUOTA Server Response Extension

An SMB_COM_NT_TRANSACTION (section 2.2.14) response for an NT_TRANSACT_QUERY_QUOTA subcommand MUST be sent by a server in reply to a client NT_TRANSACT_QUERY_QUOTA subcommand request when the request is successful.

The **TotalParameterCount** MUST be 4, and the parameter block in the server response MUST contain a 32-bit unsigned integer value indicating the length, in bytes, of the returned quota information.

The **TotalDataCount** MUST indicate the length of the Data buffer, and that buffer MUST contain the quota information defined below.



NextEntryOffset (4 bytes): An offset to the start of the subsequent entry from the start of this entry, or 0 for the final entry.

SidLength (4 bytes): The length, in bytes, of the security identifier.

ChangeTime (8 bytes): This value MUST be the time the quota was last changed, in **TIME** format.

QuotaUsed (8 bytes): The amount of quota, in bytes, used by this user. This field is formatted as a **LARGE_INTEGER**, as specified in [CIFS] section 2.4.2.

QuotaThreshold (8 bytes): The quota warning limit, in bytes, for this user. This field is formatted as a **LARGE_INTEGER**, as specified in [CIFS] section 2.4.2.

QuotaLimit (8 bytes): The quota limit, in bytes, for this user. This field is formatted as a **LARGE_INTEGER**, as specified in [CIFS] section 2.4.2.

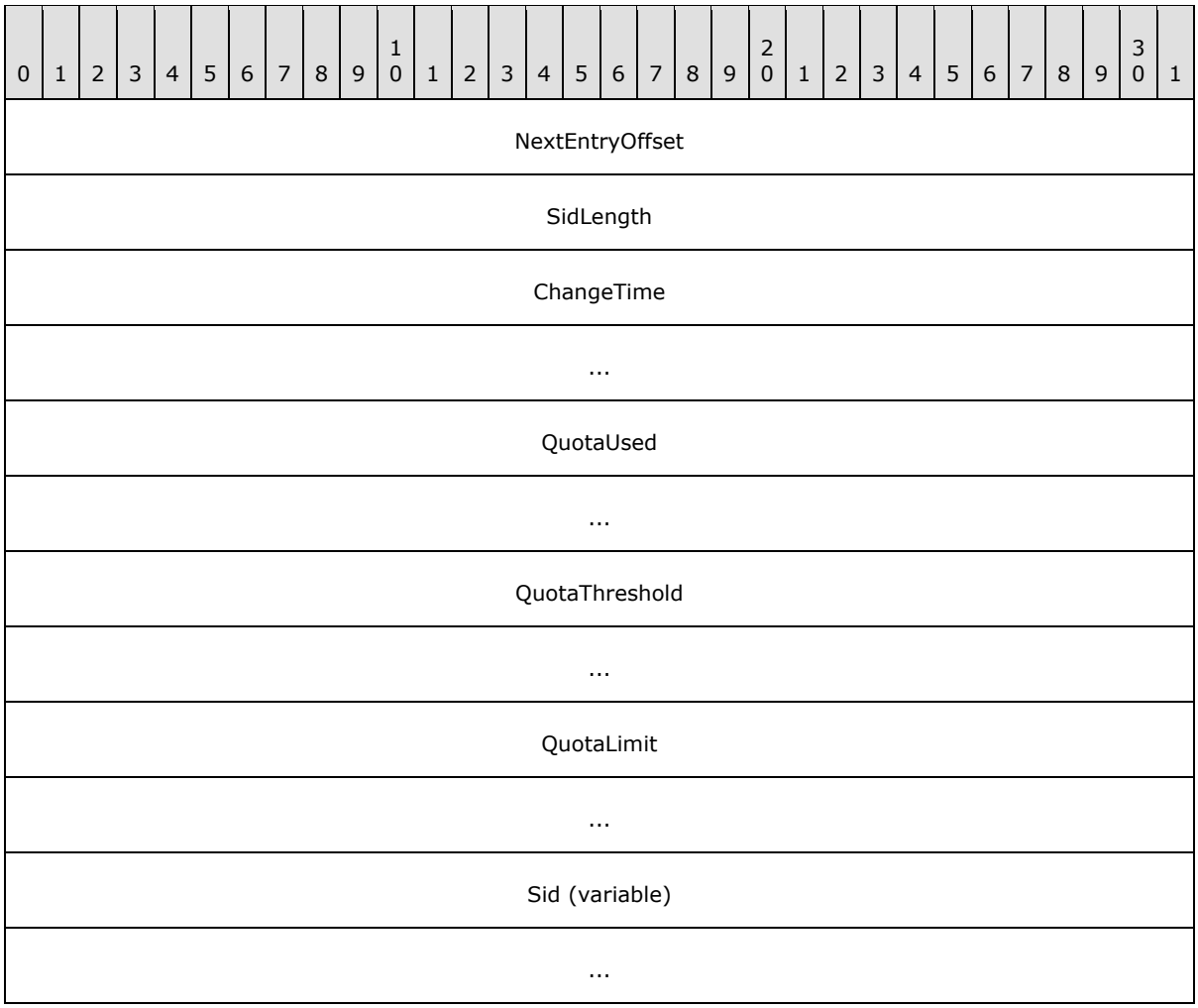
Sid (variable): The security identifier of this user. For more information, see [MS-DTYP] section **2.4.2**. Note that [CIFS] sections 4.3.4, 4.3.4.7, 4.3.5, and 4.3.5.6 use **Sid** as the field name for a search handle. In [XOPEN-SMB], the search handle field is called a findfirst_dirhandle or findnext_dirhandle. These are better field names for a search handle.

2.2.14.5 NT_TRANSACT_SET_QUOTA Client Request Extension

An SMB_COM_NT_TRANSACTION (section 2.2.14) request with an NT_TRANSACT_SET_QUOTA subcommand code is sent by a client to set quota information on a server.

The **TotalParameterCount** MUST be set to 2, and the Parameter block in the client request MUST contain the 2-byte file identifier for the target directory.

The **TotalDataCount** MUST contain the length of the Data buffer, and the Data buffer in the client request MUST contain the quota information to be applied to the system as defined below.



NextEntryOffset (4 bytes): An offset to the start of the subsequent entry from the start of this entry, or 0 for the final entry.

SidLength (4 bytes): The length, in bytes, of the security identifier.

ChangeTime (8 bytes): This value MUST be the time the quota was last changed, in **TIME** format.

QuotaUsed (8 bytes): The amount of quota, in bytes, used by this user. This field is formatted as a **LARGE_INTEGER**, as specified in [CIFS] section 2.4.2.

QuotaThreshold (8 bytes): The quota warning limit, in bytes, for this user. This field is formatted as a **LARGE_INTEGER**, as specified in [CIFS] section 2.4.2.

QuotaLimit (8 bytes): The quota limit, in bytes, for this user. This field is formatted as a **LARGE_INTEGER**, as specified in [CIFS] section 2.4.2.

Sid (variable): The security identifier of this user. For details, see [MS-DTYP] section **2.4.2**. Note that [CIFS] sections 4.3.4, 4.3.4.7, 4.3.5, and 4.3.5.6 use **Sid** as the field name for a

search handle. In [XOPEN-SMB], the search handle field is called a findfirst_dirhandle or findnext_dirhandle. These are better field names for a search handle.

2.2.14.6 NT_TRANSACT_SET_QUOTA Server Response Extension

An SMB_COM_NT_TRANSACTION (section 2.2.14) response for the NT_TRANSACT_SET_QUOTA (section 2.2.14.6) subcommand MUST be sent by a server in reply to a client NT_TRANSACT_SET_QUOTA request when the request is successful.

The server response MUST indicate if the operation was successful by setting the **Status** field of the SMB header in the response. The **Status** field MUST be set to STATUS_SUCCESS if the operation was successful, or an error code on failure. The **TotalParameterCount** and **TotalDataCount** fields MUST be set to 0.

2.2.14.7 NT_TRANSACT_IOCTL Client Request Extension

An SMB_COM_NT_TRANSACTION (section 2.2.14) request with an NT_TRANSACT_IOCTL subcommand is sent by a client to pass an IOCTL or FSCTL (file system control) command to a server. The NT_TRANSACT_IOCTL subcommand of the SMB_COM_NT_TRANSACTION request is as specified in [CIFS] section 4.6.1.

There is not a list of FSCTL and IOCTL requests that is issued by clients or handled by servers, as specified in [CIFS]. FSCTL and IOCTL control codes are implementation dependent.

For FSCTL or IOCTL requests, the target file MUST be passed in the **Fid** field on the setup words, and the control code MUST be passed in the **FunctionCode** field on the setup words. The encoding of the data block is function-code dependent. The **MaxDataCount** field in the client request MUST indicate the maximum size of output data that the client is prepared to accept in the server response.<138>

If an application running on a client requests an FSCTL or IOCTL of which the SMB implementation is unaware, the format of the data is not well known outside of the caller. The SMB protocol SHOULD pass through the request. A server that receives an unexpected FSCTL or IOCTL SHOULD fail the operation by returning STATUS_NOT_SUPPORTED.<139>

Along with the FSCTL operations specified in [MS-FSCC], there are three FSCTL operations that are specific to the extensions specified in this document.

FSCTL	Value	Description
FSCTL_SRV_ENUMERATE_SNAPSHOTS	0x00144064	Access previous versions of a file.
FSCTL_SRV_REQUEST_RESUME_KEY	0x00140078	Retrieve an opaque file reference for server-side data movement.
FSCTL_SRV_COPYCHUNK	0x001440F2	Use for server-side data movement.

2.2.14.7.1 FSCTL_SRV_ENUMERATE_SNAPSHOTS Request

This FSCTL is used to access previous file versions, as specified in section 3.2.4.11.1. The request is sent as an NT_TRANSACT_IOCTL request (section 2.2.14.7). The 32-bit **FunctionCode** for this call is 0x00144064. There is no input data for this request (**TotalDataCount** MUST be 0). The request MUST allow for at least 0x000d (13) bytes of response data (**MaxDataCount** MUST be equal to or greater than 0x000d). The response structure is specified in section 2.2.14.8.1.

2.2.14.7.2 FSCTL_SRV_REQUEST_RESUME_KEY Request

This FSCTL is used to retrieve an opaque file reference for server-side data movement operations, as specified in section 3.2.4.11.2. The request is sent as an NT_TRANSACT_IOCTL request (section 2.2.14.7). The 32-bit **FunctionCode** for this call is 0x00140078. There is no input data for this request (**TotalDataCount** MUST be set to 0). The request MUST allow for at least 0x001d (29) bytes of response data (**MaxDataCount** field MUST be 0x001d or greater). The response structure is as specified in section 2.2.14.8.2.

2.2.14.7.3 FSCTL_SRV_COPYCHUNK Request

This FSCTL is used for server-side data movement, as specified in section 3.1.6. The request is sent as an NT_TRANSACT_IOCTL request (section 2.2.14.7). The 32-bit **FunctionCode** for this call is 0x001440F2. The input data buffer contains the structure defined in the list below the table. **TotalDataCount** field MUST be equal to or greater than 0x0044 (52). The request MUST allow for at least 0x000C (12) bytes of response data. The **MaxDataCount** field MUST be equal to or greater than 0x000C (12). The response structure is as specified in section 2.2.14.8.3.

The input data format includes the following client data block encoding.

0	1	2	3	4	5	6	7	8	9	10	1	2	3	4	5	6	7	8	9	20	1	2	3	4	5	6	7	8	9	30	1
Copychunk Resume Key																															
...																															
...																															
...																															
...																															
...																															
ChunkCount																															
Unused																															
List (variable)																															
...																															

Copychunk Resume Key (24 bytes): A 24-byte server resume key for a source file in a server-side file copy operation. This MUST be an opaque file identifier returned from a previous FSCTL_SRV_REQUEST_RESUME_KEY client request.

ChunkCount (4 bytes): The number of entries in the copychunk list. MUST NOT be 0.

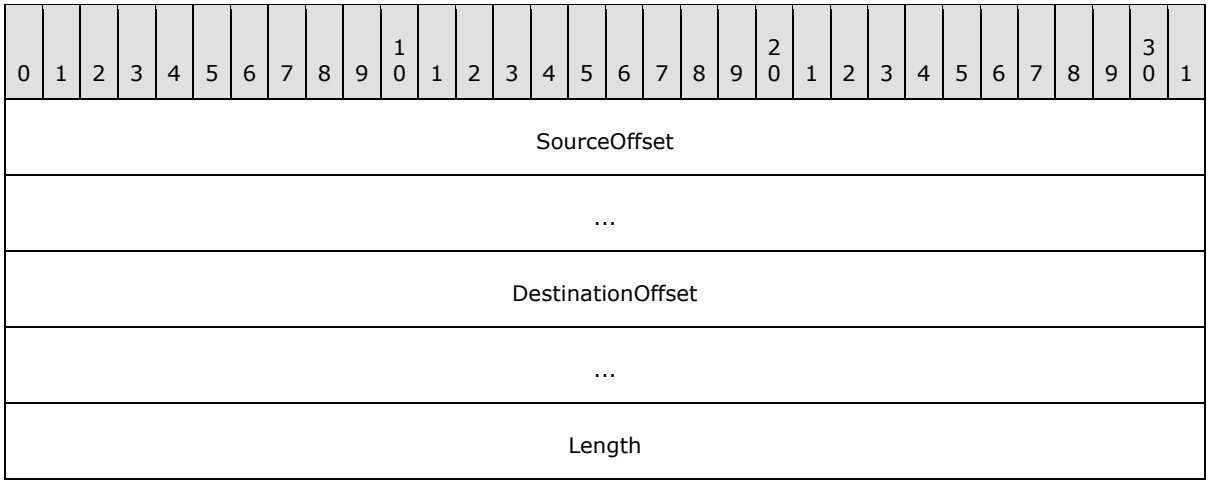
Unused (4 bytes): Reserved. This field SHOULD be set to 0 when sending the request. This field MUST be ignored by the server when the message is received.

List (variable): A concatenated list of copychunk blocks. This field is as specified in section 2.2.14.7.3.1.

The request instructs the server to attempt **ChunkCount** data movement operations, copying **Length** bytes of data from the **SourceOffset** position in the source file (specified by the opaque copychunk resume key) to the **DestinationOffset** position in the target file (specified in the SMB **Fid** field passed to the NT_TRANSACT_IOCTL (section 2.2.14.7) client request).

2.2.14.7.3.1 FSCTL_SRV_COPYCHUNK Request List

The List packet is a concatenated list of copychunk blocks. Each such block is defined as follows.



SourceOffset (8 bytes): The offset, in bytes, in the source file to copy from. This field is formatted as a **LARGE_INTEGER**, as specified in [CIFS] section 2.4.2.

DestinationOffset (8 bytes): The offset, in bytes, in the target file to copy to. This field is formatted as a **LARGE_INTEGER**, as specified in [CIFS] section 2.4.2.

Length (4 bytes): The number of bytes to copy from the source file to the target file.

2.2.14.8 NT_TRANSACT_IOCTL Server Response Extensions

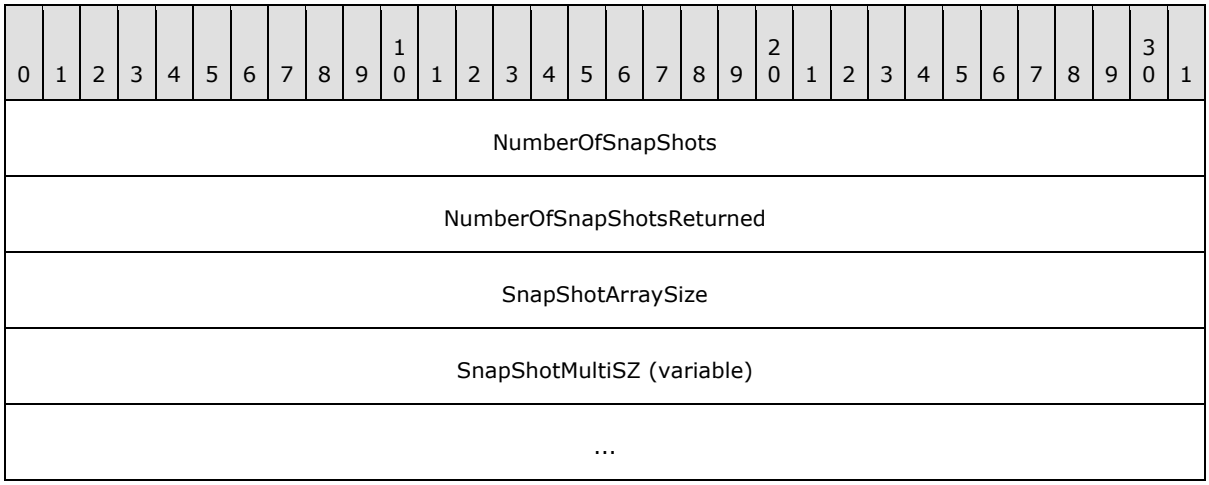
An SMB_COM_NT_TRANSACTION (section 2.2.14) response for an NT_TRANSACT_IOCTL (section 2.2.14.7) subcommand MUST be sent by a server in reply to a client NT_TRANSACT_IOCTL request when the request is successful. The server response to the client NT_TRANSACT_IOCTL request is specified in [CIFS] section 4.6.1.

The complete response format for FSCTL and IOCTL operations is not specified in [CIFS]. The **TotalDataCount** field in the response indicates the size of the returned data, and the format of the **Data** field in the data block is control code dependent.<140>

2.2.14.8.1 FSCTL_SRV_ENUMERATE_SNAPSHOTS Response

This is the response format for a successful NT_TRANSACT_IOCTL request executed with an FSCTL_SRV_ENUMERATE_SNAPSHOTS **FunctionCode**, as specified in section 2.2.14.7.1. There are

no output parameters (**TotalParameterCount** MUST be 0), but the output data buffer MUST contain the structure with the following server data block encoding, and **TotalDataCount** MUST indicate the length of this structure.



NumberOfSnapShots (4 bytes): This value MUST be the number of snapshots for the volume.

NumberOfSnapShotsReturned (4 bytes): This value MUST be the number of snapshots being returned in this packet.

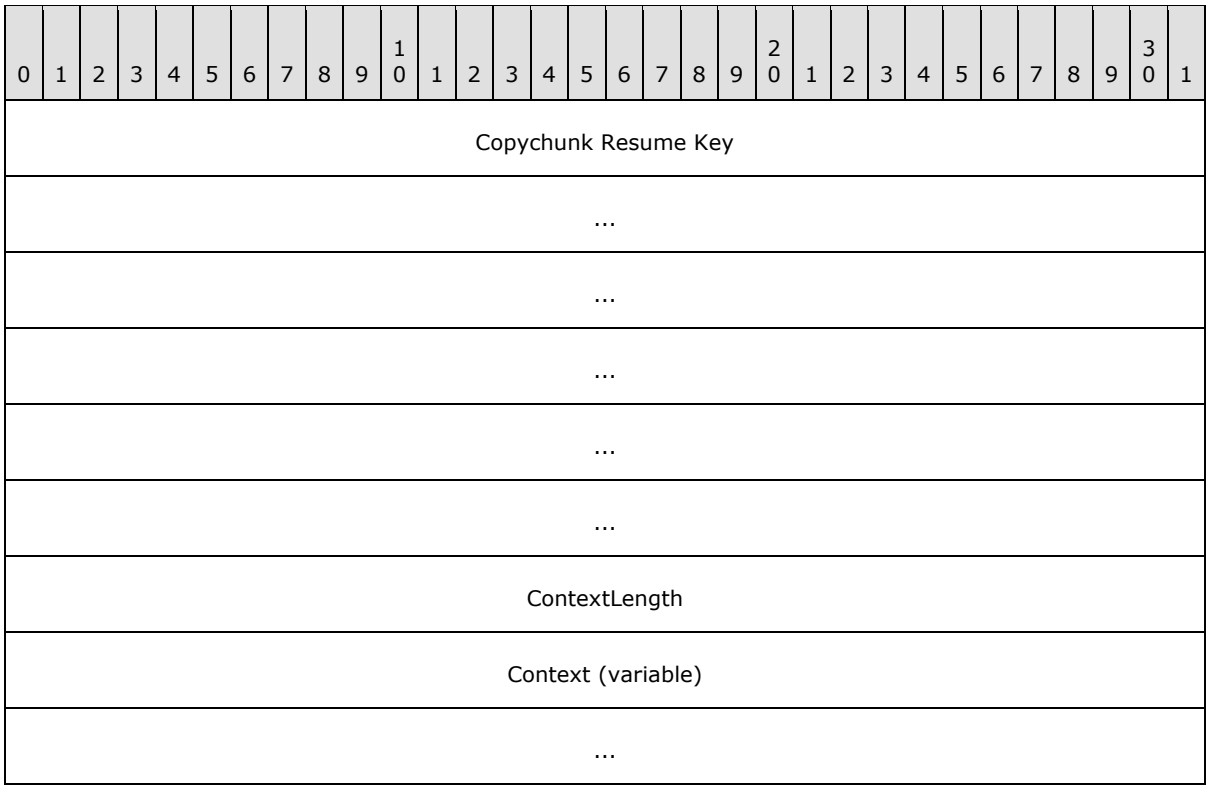
SnapShotArraySize (4 bytes): The size, in bytes, needed for the array.

SnapShotMultiSZ (variable): A concatenated set of snapshot names. Each snapshot name MUST be formatted as a null-terminated array of 16-bit Unicode characters. The concatenated list MUST be terminated by two 16-bit Unicode NULL characters.

Each string in the array MUST be an array of 16-bit Unicode characters in the format of an @GMT token; that is, @GMT-YYYY.MM.DD-HH.MM.SS. If all of the snapshots cannot fit into the callers' buffer, based on the **MaxDataCount** of the request, **NumberOfSnapShotsReturned** MUST be less than **NumberOfSnapShots**. **SnapShotArraySize** MUST contain the size required to receive the entire array, not including the size of the enumeration structure.

2.2.14.8.2 FSCTL_SRV_REQUEST_RESUME_KEY Response

This is the response format for a successful NT_TRANSACT_IOCTL request executed with the FSCTL_SRV_REQUEST_RESUME_KEY **FunctionCode**, as specified in FSCTL_SRV_REQUEST_RESUME_KEY Request (section 2.2.14.7.2). There are no output parameters (**TotalParameterCount** MUST be 0), but the output data buffer MUST contain the structure with the following server data block encoding; and **TotalDataCount** MUST indicate the length of this structure.



Copychunk Resume Key (24 bytes): A 24-byte resume key generated by the SMB server that can be subsequently used by the client to uniquely identify the open source file in a FSCTL_SRV_COPYCHUNK request. The resume key MUST be treated as a 24-byte opaque structure. The SMB client that receives the 24-byte copychunk resume key MUST NOT attach any interpretation to this key and MUST treat it as an opaque value. For more information, see section 1.8.4.

ContextLength (4 bytes): The length, in bytes, of the context information. This field is unused. A server SHOULD set this field to 0 when sending a response. This field MUST be ignored by the client when the message is received.<141>

Context (variable): The context extended information.

2.2.14.8.3 FSCTL_SRV_COPYCHUNK Response

This is the response format for an NT_TRANSACT_IOCTL request executed with the FSCTL_SRV_COPYCHUNK **FunctionCode**, as specified in FSCTL_SRV_COPYCHUNK Request. There are no output parameters (**TotalParameterCount** MUST be 0), but the output data MUST contain the following structure of server data block encoding fields, and **MaxDataCount** MUST indicate the length of this buffer.

0	1	2	3	4	5	6	7	8	9	10	1	2	3	4	5	6	7	8	9	20	1	2	3	4	5	6	7	8	9	30	1
ChunksWritten																															
ChunkBytesWritten																															
TotalBytesWritten																															

ChunksWritten (4 bytes): This value MUST be the number of chunks successfully processed by the server.

ChunkBytesWritten (4 bytes): This value MUST be the number of bytes written to the target file.

TotalBytesWritten (4 bytes): This value MUST be the total number of bytes written to the target file.

2.2.15 Extended Attribute Encoding Extensions

The list of file attributes valid in 32-bit attribute values, as specified in [CIFS] section 3.11, has been extended. The following table lists all possible values for completeness. Unless otherwise noted, any combination of these values is acceptable. **Note** [CIFS] defines the values in little-endian format; they are shown here in packet format.

0	1	2	3	4	5	6	7	8	9	10	1	2	3	4	5	6	7	8	9	20	1	2	3	4	5	6	7	8	9	30	1
X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	EN	NI	OF	CO	RP	SP	TM	NO	X	AR	DI	X	SY	HI	RO

RO: (ATTR_READONLY) File is read only. Applications cannot write or delete the file.

HI: (ATTR_HIDDEN) File is hidden. It is not to be included in an ordinary directory enumeration.

SY: (ATTR_SYSTEM) File is part of or is used exclusively by the operating system.

X: Unused bit fields. SHOULD be set to 0 when sending, and MUST be ignored when the message is received.

DI: (ATTR_DIRECTORY) File is a directory.

AR: (ATTR_ARCHIVE) File has not been archived since it was last modified.

NO: (ATTR_NORMAL) File has no other attributes set. This value is valid only when used alone.

TM: (ATTR_TEMPORARY) File is temporary.

SP: (ATTR_SPARSE) File is a sparse file.

RP: (ATTR_REPARSE_POINT) File or directory has an associated reparse point.

CO: (ATTR_COMPRESSED) File is compressed on the disk. This does not affect how it is transferred over the network.

OF: (ATTR_OFFLINE) File data is not available immediately. The attribute indicates the file has been moved to offline storage.

NI: (ATTR_NOT_CONTENT_INDEXED) File or directory SHOULD NOT be indexed by a content indexing service.

EN: (ATTR_ENCRYPTED) File or directory is encrypted. For a file, this means that all data in the file is encrypted. For a directory, this means that encryption is the default for newly created files and subdirectories.

2.2.16 File System Attribute Extensions

The list of file system attributes, as specified in [CIFS] section 4.1.6.6, has been extended. For completeness, the following table lists attributes and their symbolic constants. Unless otherwise noted, any combination of the bits below is valid. Any bit that is not listed below is considered reserved; the sender SHOULD set it to 0, and the receiver MUST ignore it.

0	1	2	3	4	5	6	7	8	9	10	1	2	3	4	5	6	7	8	9	20	1	2	3	4	5	6	7	8	9	30	1
X	X	X	X	X	X	X	X	X	X	X	X	RO	NS	EN	OI	VC	X	X	X	X	X	X	RS	RP	SP	VQ	FC	PA	UC	PN	CS

CS: (FILE_CASE_SENSITIVE_SEARCH) File system supports case-sensitive file names.

PN: (FILE_CASE_PRESERVED_NAMES) File system preserves the case of file names when it stores the name on disk.

UC: (FILE_UNICODE_ON_DISK) File system supports Unicode in file names.

PA: (FILE_PERSISTENT_ACLS) File system preserves and enforces access control lists.

Note In [CIFS] section 4.1.6.6, this attribute was listed incorrectly as FILE_PERSISTENT_ACLS.

FC: (FILE_FILE_COMPRESSION) File system supports file-based compression. This flag is incompatible with **FILE_VOLUME_IS_COMPRESSED**. This flag does not affect how data is transferred over the network.

VQ: (FILE_VOLUME_QUOTAS) File system supports per-user quotas.

SP: (FILE_SUPPORTS_SPARSE_FILES) File system supports sparse files.

RP: (FILE_SUPPORTS_REPARSE_POINTS) File system supports reparse points.

RS: (FILE_SUPPORTS_REMOTE_STORAGE)

X: (Reserved) These bits fields SHOULD be set to 0 when sending and MUST be ignored when the message is received.

VC: (FILE_VOLUME_IS_COMPRESSED) Volume is a compressed volume. This flag is incompatible with **FILE_FILE_COMPRESSION**. This does not affect how data is transferred over the network.

OI: (FILE_SUPPORTS_OBJECT_IDS) File system supports object identifiers.

EN: (FILE_SUPPORTS_ENCRYPTION) File system supports encryption.

NS: (FILE_NAMED_STREAMS) File system supports multiple named data streams for a file.

RO: (FILE_READ_ONLY_VOLUME) Specified volume is read only.

2.2.17 32-Bit Status Codes

The following is a list of 32-bit status codes required to implement these extensions, their associated values, and a description of what they represent.<142>

Return value/code	Description
0x00000000 STATUS_SUCCESS	The client request is successful.
0xC000000D STATUS_INVALID_PARAMETER	The parameter specified in the request is not valid.
0xC0000016 STATUS_MORE_PROCESSING_REQUIRED	The specified I/O request packet (IRP) cannot be disposed of because the I/O operation is not complete.
0xC0000022 STATUS_ACCESS_DENIED	The client did not have the required permission needed for the operation.
0xC000003A STATUS_OBJECT_PATH_NOT_FOUND	The path to the directory specified was not found. This error is also returned on a create request if the operation requires creating more than one new directory level for the path specified.
0xC00000BB STATUS_NOT_SUPPORTED	The client request is not supported.
0xC00000C9 STATUS_NETWORK_NAME_DELETED	The network name specified by the client has been deleted on the server. This error is returned if the client specifies an incorrect Tid or the share on the server represented by the Tid was deleted.
0xC0000203 STATUS_USER_SESSION_DELETED	The user session specified by the client has been deleted on the server. This error is returned by the server if the client sends an incorrect Uid.
0xC000035C STATUS_NETWORK_SESSION_EXPIRED	The client's session has expired, therefore the client MUST re-authenticate to continue accessing remote resources.
0xC0002001 STATUS_INVALID_SMB	An invalid SMB client request is received by the server.
0xC0002005 STATUS_SMB_BAD_TID	The client request received by the server contains an invalid Tid value.
0xC0002016 STATUS_SMB_BAD_COMMAND	The client request received by the server contains an unknown SMB command code.
0xC000205A	The client has requested too many Uid values from the

Return value/code	Description
STATUS_SMB_TOO_MANY_UIDS	server or the client already has a an SMB session setup with this Uid value.
0xC000205B STATUS_SMB_BAD_UID	The client request to the server contains an invalid Tid value.
0xC00020FB STATUS_SMB_USE_STANDARD	The client request received by the server is for a non-standard SMB operation (for example, an SMB_COM_READ_MPX request on a non-disk share). The client SHOULD send another request with a different SMB command to perform this operation.

3 Protocol Details

SMB operates between an initiator (client) and a responder (server), as specified in [CIFS]. Details common to both endpoints are covered first, and then any additional client-specific or server-specific details are covered.

3.1 Common Details

3.1.1 Abstract Data Model

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

3.1.1.1 Global

The following additional parameters are required beyond what is required in the CIFS Protocol, as specified in [CIFS].

MessageSigningPolicy: A state that determines if this node signs messages. This has four possible values:

- Message signing is required. Connections to parties who do not use signing MUST be disconnected.
- Message signing is enabled. If the other party supports signing, it MUST be used.
- Message signing is disabled unless the other party requires it. If the other party requires message signing, it MUST be used. Otherwise, message signing MUST NOT be used.
- Message signing is disabled. Message signing MUST NOT be used.

SupportsExtendedSecurity: A flag that indicates if this node supports authentication, as specified in [RFC4178], for selecting the authentication protocol.

3.1.1.2 Per SMB Connection

The following parameters are required in addition to those parameters required by the CIFS Protocol, as specified in [CIFS]:

- **IsSigningActive:** A **BOOLEAN** that indicates if message signing is active for this SMB connection.
- **ConnectionSigningSessionKey:** A 16-byte array containing the session key that is being used for signing packets, if signing is active.
- **ConnectionSigningNTLMChallengeResponse:** A variable-length byte array containing the NTLM challenge response to use for signing, if signing is active.
- **ConnectionSigningUseNTLMChallengeResponse:** A Boolean that is used to determine if ConnectionSigningNTLMChallengeResponse is used in the signature calculation.

Note that the above conceptual data can be implemented using a variety of techniques. This specification places no constraints on how these logical data types are realized in an actual implementation.

3.1.2 Timers

There are no new timers other than what is required by the CIFS Protocol, as specified in [CIFS].

3.1.3 Initialization

Values for MessageSigningPolicy MUST be set based on system policy, and MUST be one of the possible values listed in Global (section 3.1.1.1). The value of this is not constrained by the values of any other policies.

SupportsExtendedSecurity MUST be set based on system policy. The value of this is not constrained by the values of any other policies.<143>

When an SMB connection is established, the following values MUST be initialized:

- IsSigningActive is set to FALSE.
- ConnectionSigningUseNTLMChallengeResponse is set to FALSE.
- ConnectionSigningNTLMChallengeResponse is a zero-length array.
- ConnectionSigningSessionKey is zeroed.

3.1.4 Higher-Layer Triggered Events

3.1.4.1 Sending Any Message

If a message is being sent and IsSigningActive is TRUE, the message MUST be signed. This logic MUST be applied for messages sent in response to any of the higher-layer actions and in response to any message sequencing rules.

The client or server that is sending the message MUST provide the 32-bit sequence number for this message, as specified in sections 3.2.4.1 and 3.3.4.1. To generate the signature, the 8-byte **SecuritySignature** field in the SMB header of the message being signed is zeroed. Then the 32-bit sequence number is copied into the first 4 bytes of the **SecuritySignature** field. The MD5 algorithm, as specified in [RFC1321], MUST be used to generate a hash of the SMB message (from the start of the SMB header), which is defined as follows.

```
IF ConnectionSigningUseNTLMChallengeResponse THEN
   CALL MD5Init( md5context )
   CALL MD5Update( md5context, ConnectionSigningSessionKey )
   CALL MD5Update( md5context,
                   ConnectionSigningNTLMChallengeResponse )
   CALL MD5Update( md5context, SMB message )
   CALL MD5Final( digest, md5context )
ELSE
   CALL MD5Init( md5context )
   CALL MD5Update( md5context, ConnectionSigningSessionKey )
   CALL MD5Update( md5context, SMB message )
   CALL MD5Final( digest, md5context )
END IF
```

```
SET signature TO first 8 bytes of digest
```

The resulting 8-byte signature MUST be copied into the **SecuritySignature** field of the SMB header.

3.1.5 Message Processing Events and Sequencing Rules

3.1.5.1 Receiving Any Message

If a message is being received and IsSigningActive is TRUE, the signature of the message MUST be verified (unless otherwise noted) in the client receive (see section 3.2.5.1) or the server receive (see section 3.3.5.1). This logic MUST be applied for any messages received, as defined in the message sequencing rules.

The client or server that is receiving the message MUST save a temporary copy of the **SecuritySignature** field from the SMB header of the received message. To test the signature, the 8-byte **SecuritySignature** field in the SMB header of the message received is zeroed. Then the 32-bit sequence number from the received message is copied into the first 4 bytes of the **SecuritySignature** field. The MD5 algorithm, as specified in [RFC1321], MUST be used to generate a hash of the SMB message (from the start of the SMB header), which is defined as follows.

```
IF ConnectionSigningUseNTLMChallengeResponse THEN
   CALL MD5Init( md5context )
   CALL MD5Update( md5context, ConnectionSigningSessionKey )
   CALL MD5Update( md5context,
                   ConnectionSigningNTLMChallengeResponse )
   CALL MD5Update( md5context, SMB message )
   CALL MD5Final( digest, md5context )
ELSE
   CALL MD5Init( md5context )
   CALL MD5Update( md5context, ConnectionSigningSessionKey )
   CALL MD5Update( md5context, SMB message )
   CALL MD5Final( digest, md5context )
END IF
SET signature TO first 8 bytes of digest
```

The resulting 8t-byte signature is compared with the original value of the **SecuritySignature** field from the SMB header. If the signature received with the message does not match the signature calculated, the message MUST be discarded, and no further processing on it is done. The receiver MAY also terminate the connection by disconnecting the underlying transport connection and cleaning up any state associated with the connection.<144>

3.1.6 Timer Events

There are no new timers other than what is required by the CIFS Protocol, as specified in [CIFS].

3.1.7 Other Local Events

There are no local events other than what is required by the CIFS Protocol, as specified in [CIFS].

3.2 Client Details

3.2.1 Abstract Data Model

No additional data is required beyond what is required by the CIFS Protocol, as specified in [CIFS], except for those included in the common section (see section 3.1.1) and the following additional parameters.

3.2.1.1 Global

There is no new global state.

3.2.1.2 Per SMB Connection

ServerGUID: A universal **GUID** generated by the server to uniquely identify this server.

UsesSharePasswords: A Boolean that determines if the target server uses share passwords instead of user accounts.

ServerSigningState: A value that indicates the signing policy of the server. This value can be Disabled, Enabled, or Required.

ClientNextSendSequenceNumber: A sequence number for the next signed request being sent.

ClientResponseSequenceNumber: A list of the expected sequence numbers for the responses of outstanding signed requests, indexed by message ID (Mid).

NTLMEncryptionKey: A byte array containing the encryption key received from the server during a non-extended security negotiate for use in implicit NTLM authentication and remembered for authentication.

GSSNegotiateToken: A byte array containing the token received during an extended security negotiation and remembered for authentication.

ServerCapabilities: The capabilities of the server, as specified in section 2.2.3. The capabilities indirectly reflect the negotiated dialect for this connection.

NegotiateSent: A Boolean that indicates if a negotiation packet has been sent for this connection.

MaxBufferSize: The negotiated maximum size, in bytes, for SMB messages sent between the client and the server.

SessionList: A list of authenticated sessions that has been established on this SMB connection and the associated state. It MUST be possible to look up entries by either the Uid or the security context of the user that established the session.

TreeConnectList: A list of the tree connects over this SMB connection established to shares on the target server, containing the Tid (as specified in [CIFS] section 3.1.3) for that tree connect. It MUST be possible to look up entries by either Tid or by share name. Each tree connect entry contains a MaximalShareAccessRights and GuestMaximalAccessRights value, as specified in the ACCESS_MASK format, as specified in [CIFS] section 3.7. The MaximalShareAccessRights and GuestMaximalAccessRights values are returned in the SMB_COM_TREE_CONNECT_ANDX server response, as specified in section 2.2.7.

3.2.1.3 Per SMB Session

SessionState : A session can be in one of three states:

- InProgress – A session setup is in progress for this session.
- Valid – The session is valid and a session key and UID are available for this session.
- Expired – The Kerberos ticket for this session has expired and the session needs to be re-established.

UserSecurityContext: The security context of the user which established the session.

SessionUid : The 2-byte UID for this session. The UID is returned by the server in the SMB header of session setup response. All subsequent SMB requests for this user on this connection must use this UID.

SessionKey : A 16-byte session key for this session. If SMB signing is active on the connection, the session key from the first non-null, non-guest session is used for signing all traffic on the SMB connection.

3.2.1.4 Per SMB Tree Connection

ShareName: The share name corresponding to this tree connection.

TreeId: The tree-ID identifying this tree connection as returned by the server in the header of the SMB_COM_TREE_CONNECT_ANDX server response (section 2.2.7), as specified in [CIFS] section 3.1.3.

MaximalShareAccessRights: The MaximalShareAccessRights value as returned in the SMB_COM_TREE_CONNECT_ANDX server response (section 2.2.7).

GuestMaximalShareAccessRights: The GuestMaximalShareAccessRights value as returned in the SMB_COM_TREE_CONNECT_ANDX server response (section 2.2.7).

3.2.2 Timers

There are no new client-side timers other than what is required by the CIFS Protocol, as specified in [CIFS].

3.2.3 Initialization

There following parameters need to be initialized in addition to what is required in the CIFS Protocol, as specified in [CIFS].

When an SMB connection is established, the following values MUST be initialized:

- ServerGUID is set to 0.
- UsesSharePasswords is set to FALSE.
- ServerSigningState is Disabled.
- ClientNextSendSequenceNumber is set to 2.
- ClientResponseSequenceNumber is an empty list.

- GSSNegotiateToken is set to an empty array.
- NTLMEncryptionKey is set to an empty array.
- ServerCapabilities is set to 0.
- NegotiateSent is set to FALSE.
- MaxBufferSize is set to 0.
- SessionList is set to an empty list.
- TreeConnectList is set to an empty list.

3.2.4 Higher-Layer Triggered Events

3.2.4.1 Sending Any Message

If a message is being sent, and signing is active for the connection, the message MUST be signed as specified in section 3.1.4.1. As defined in that section, the logic below MUST be used for all sends after IsSigningActive is set to TRUE.

The client is responsible for providing the appropriate sequence number for signature calculation. The sequence number for the next request is stored in ClientNextSendSequenceNumber. The client MUST remember the appropriate sequence number for a response, and does so by inserting it into the ClientResponseSequenceNumber table with the Mid that identifies the request/response pair. (Mid is as specified in [CIFS] section 3.1.5). After signing the message with ClientNextSendSequenceNumber, the following steps MUST be taken.

```
IF request command EQUALS SMB_COM_NT_CANCEL THEN
   INCREMENT ClientNextSendSequenceNumber
ELSE IF request has no response THEN
   INCREMENT ClientNextSendSequenceNumber BY 2
ELSE
   SET ClientResponseSequenceNumber[Mid] TO
      ClientNextSendSequenceNumber + 1
   INCREMENT ClientNextSendSequenceNumber BY 2
END IF
```

The client MUST ensure that packets are sent to the server in the same order they are signed, to guarantee that the sequence numbers match during server validation.

3.2.4.2 Application Requests Connecting to a Server

The higher-layer application MUST provide the name of the server to connect to, and the name of the share on the server that it is connecting to, and the authentication context of the user who is attempting the connection, and a flag indicating whether the path is a DFS path. It also has the option of specifying the SMB transport on which it wants the connection to be attempted. The client MUST follow the steps as described in the following flowchart. The request to connect to a server can be either explicit (for example, the application requests an SMB connection to \\server\share) or implicit (for example, the application requests to open the file \\server\share\file.txt, which implies that an SMB connection to \\server\share is being established). In either case, the steps described

below are followed. The only difference is that for the implicit case, the error returned in the failure case MUST be returned as the error of the operation that caused the implicit connect attempt.

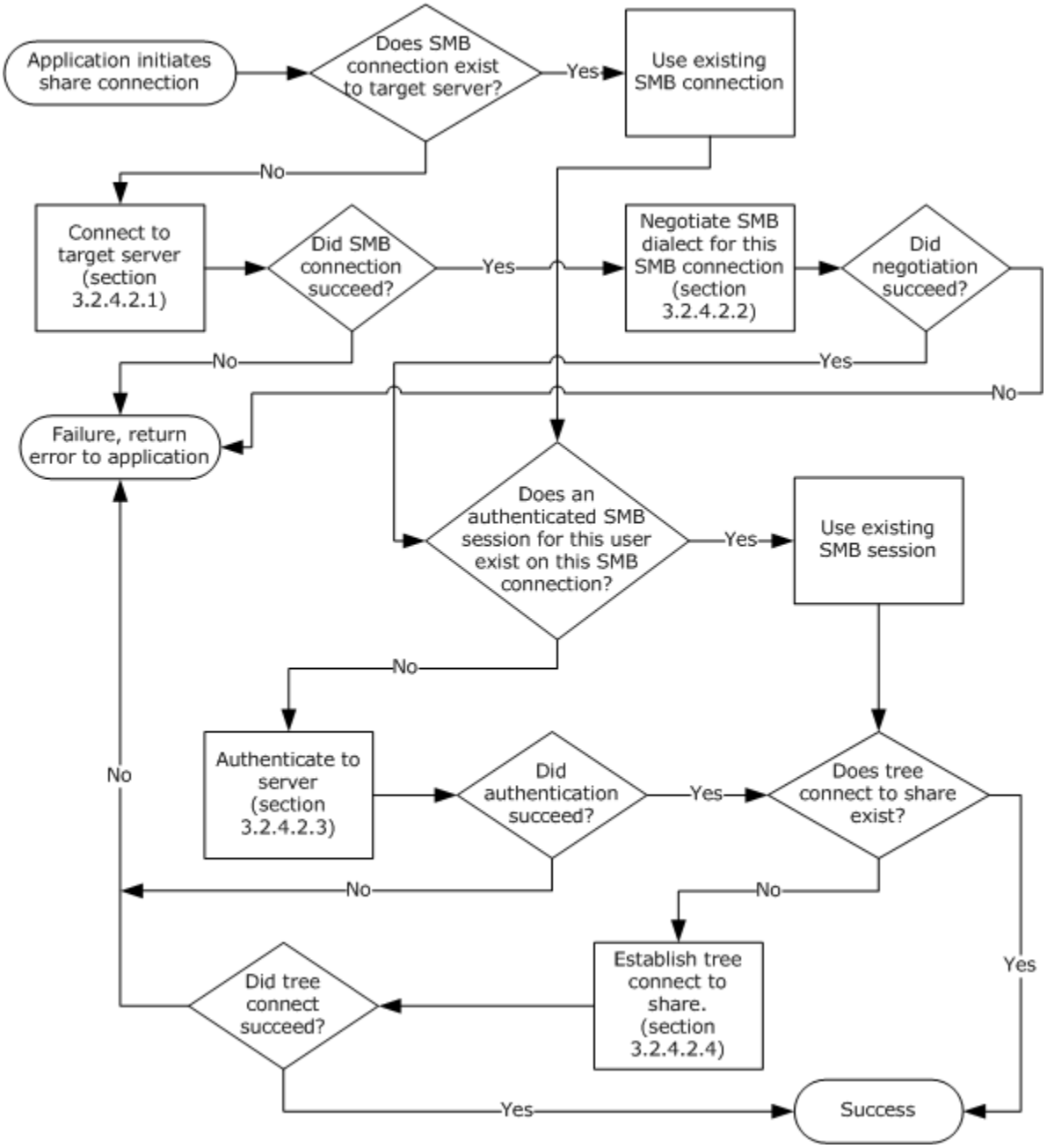


Figure 1: Application that connects to a share on a server

To complete an SMB connection to a share, the client MUST have established an SMB transport connection, an authenticated SMB session for the user initiating the call, and a tree connection to the target share.

3.2.4.2.1 Connection Establishment

If there is no existing SMB connection, a new SMB connection MUST be established. Executing this operation requires the name of the server being connected to, and the name of the SMB transport to use, if specified. The client MAY attempt to initiate the SMB connection on all SMB transports it supports, most commonly NetBIOS over TCP (as specified in [CIFS] section 2.6) and the other SMB transports (as specified in section 2.1). The client MAY choose to prioritize the SMB transport order and try each SMB transport sequentially, or try to connect on all SMB transports and select one using any implementation-specific heuristic. The client MAY accept information from the calling application that specifies what SMB transport to use, and attempt to use the transport specified.<145>

If all connection attempts fail, the connection establishment is failed and an appropriate error is returned, which is passed back to the calling application, as described earlier.

3.2.4.2.2 Dialect Negotiation

If NegotiateSent is FALSE, the client MUST negotiate a dialect (as specified in [CIFS] section 2.5) before progressing to other operations on the connection or allowing other share connection attempts to use the same connection.

In addition to the steps specified in the CIFS Protocol (as specified in [CIFS]), the following steps MUST be followed when generating the SMB_COM_NEGOTIATE request. When the SMB_COM_NEGOTIATE request is sent to the server, NegotiateSent is set to TRUE:

- Extended Security

  If the SupportsExtendedSecurity is TRUE, the client MUST set the **SMB_FLAGS2_EXTENDED_SECURITY** field in the SMB header of the SMB_COM_NEGOTIATE request to TRUE.

- Require Message Signing

  If the MessageSigningPolicy of the client is 1 (message signing required), the client SHOULD set the SMB_FLAGS2_SMB_SECURITY_SIGNATURE_REQUIRED bit in the **Flags2** field of the SMB header to indicate that the client MUST refuse to connect if signing is not used.<146>

3.2.4.2.3 User Authentication

If UsesSharePasswords is FALSE, and there is no entry for the security context of the user establishing the connection to the share in the SessionList, the client MUST establish an authenticated session for the user initiating the connection to the share. If UsesSharePasswords is TRUE, share passwords are being used. Share passwords are legacy and specified in [CIFS] sections 4.1.4 and 10.1. If there is already a session in the SessionList for this user, the existing session SHOULD be reused.

The client creates an SMB_COM_SESSION_SETUP_ANDX request (section 2.2.4), as specified in [CIFS] section 4.1.2. The changes from authentication, as specified in [CIFS] and in the extensions in this document, are defined as follows:

- Signing

  The client MUST set SMB_FLAGS2_SMB_SECURITY_SIGNATURE to TRUE on the SMB header if the combination of the client's MessageSigningPolicy and the connection's ServerSigningState results in Signed on the chart below. Setting this value indicates to the server that signing is

wanted. ServerSigningState was initialized on the processing of a negotiation response, as specified in section 3.2.5.2.

If the result is Blocked in the following table, the underlying transport connection must be closed.

Server Signing State \ Client Signing State	Required	Enabled	Disabled unless Required	Disabled
Required	Message Signed	Message Signed	Message Signed	Blocked
Enabled	Message Signed	Message Signed	Message Unsigned	Message Unsigned
Disabled unless required	Message Signed	Message Unsigned	Message Unsigned	Message Unsigned
Disabled	Blocked	Message Unsigned	Message Unsigned	Message Unsigned

- Extended Security

  If the CAP_EXTENDED_SECURITY bit in ServerCapabilities is set, the SMB_COM_SESSION_SETUP_ANDX command MUST be generated, as defined here. Otherwise, the client MUST generate the authentication portion, as described in section Implicit NTLM.

  The client MAY do one of the following:

  - Pass the GSSNegotiateToken (if valid) to the configured GSS authentication mechanism to obtain a GSS output token for the authentication protocol exchange.

    -OR-

  - Choose to ignore the GSSNegotiateToken received from the server, and give an empty input token to the configured GSS authentication protocol to obtain a GSS output token for the authentication protocol exchange.

    In either case, it initializes the GSS authentication protocol with the MutualAuth and Delegate options.<147>

  The client creates an SSMB_COM_SESSION_SETUP_ANDX request (section 2.2.4) message. The client MUST set CAP_EXTENDED_SECURITY in the **Capabilities** field, SMB_FLAGS2_EXTENDED_SECURITY in the SMB header **Flags2** field, the **SecurityBlob** field to the output token generated above, and the **SecurityBlobLength** to the token length. The **Uid** field in the SMB header MUST be set to 0.

- Implicit NTLM

  If the CAP_EXTENDED_SECURITY bit in ServerCapabilities is not set, the SMB_COM_SESSION_SETUP_ANDX request MUST be generated, as defined here. This is not new to the extension, but is included for completeness.

The server MUST construct an NTLM CHALLENGE_MESSAGE, as specified in [MS-NLMP]. The CHALLENGE_MESSAGE is used by the server to challenge the client to prove its identity. The following parameters MUST be set to the following values.

- The **Signature** field is set to an 8-byte character array that MUST contain the ASCII string ('N', 'T', 'L', 'M', 'S', 'S', 'P', '\0').
- The **MessageType** field is set to 2.
- The **TargetName** field is set as follows:
  - Length is the length of the server's name.
  - MaximumLength is set to 0.
  - Buffer is set to 47 (the offset of the first byte of the payload).
- NegotiateFlags are set to NTLMSSP_NEGOTIATE_NTLM|NTLMSSP_NEGOTIATE_OEM.
- ServerChallenge is set to NTLMEncryptionKey.
- The **Reserved** field is set to 0.
- The **TargetInfo** field is set as follows:
  - Length is set to 0.
  - MaximumLength is set to 0.
  - Buffer is set to 0.
- The Payload, starting at offset 47 from the start of the NTLM CHALLENGE_MESSAGE, is set to the server's name as a null-terminated array of 16-bit Unicode characters.

The client creates the appropriate AUTHENTICATE_MESSAGE to respond to the CHALLENGE_MESSAGE, as specified in [MS-NLMP].

The client MUST build an SMB_COM_SESSION_SETUP_ANDX , as specified in [CIFS] section 4.1.2. The following fields MUST be set to these values:

- The **CaseSensitivePasswordLength** field is set to the Length of the **NtChallengeResponse** from the AUTHENTICATE_MESSAGE.
- The **CaseSensitivePassword** field is set to the value of the **NtChallengeResponse** field from the AUTHENTICATE_MESSAGE.
- The **CaseInsensitivePasswordLength** field is set to the Length of the **LmChallengeResponse** from the AUTHENTICATE_MESSAGE.
- The **CaseInsensitivePassword** field is set to the value of the **LmChallengeResponse** field from the AUTHENTICATE_MESSAGE.
- The **AccountName** field is set to the value of the **UserName** field from the AUTHENTICATE_MESSAGE.
- The **PrimaryDomain** field is set to the value of the **DomainName** field from the AUTHENTICATE_MESSAGE.

The client sends the SMB_COM_SESSION_SETUP_ANDX to the server.

3.2.4.2.3.1 Sequence Diagram

For the user to get successfully authenticated and establish a session, the client MUST follow a security negotiation scheme that MAY involve multiple round trips of SMB_COM_SESSION_SETUP_ANDX request and response. In each round trip, the server and client exchange security tokens. For an explanation of how this exchange of security tokens MUST continue until either the client or the server determines that authentication has failed, or both sides decide that authentication is complete, see [RFC4178] section 1.3.2, "SPNEGO Synopsis". Should authentication fail, the client drops the connection and indicates the error (see diagram for details). Should the authentication succeed, the application protocol can be assured of the identity of the participants as far as the supporting authentication protocol can accomplish.

In the sequence diagram that follows, requests with straight line arrows stand for the requests the client MUST send. Responses with straight line arrows stand for the responses the server MUST send. Requests with dotted line arrows stand for the requests the client MAY send. And responses with dotted line arrows stand for the responses the server MAY send.

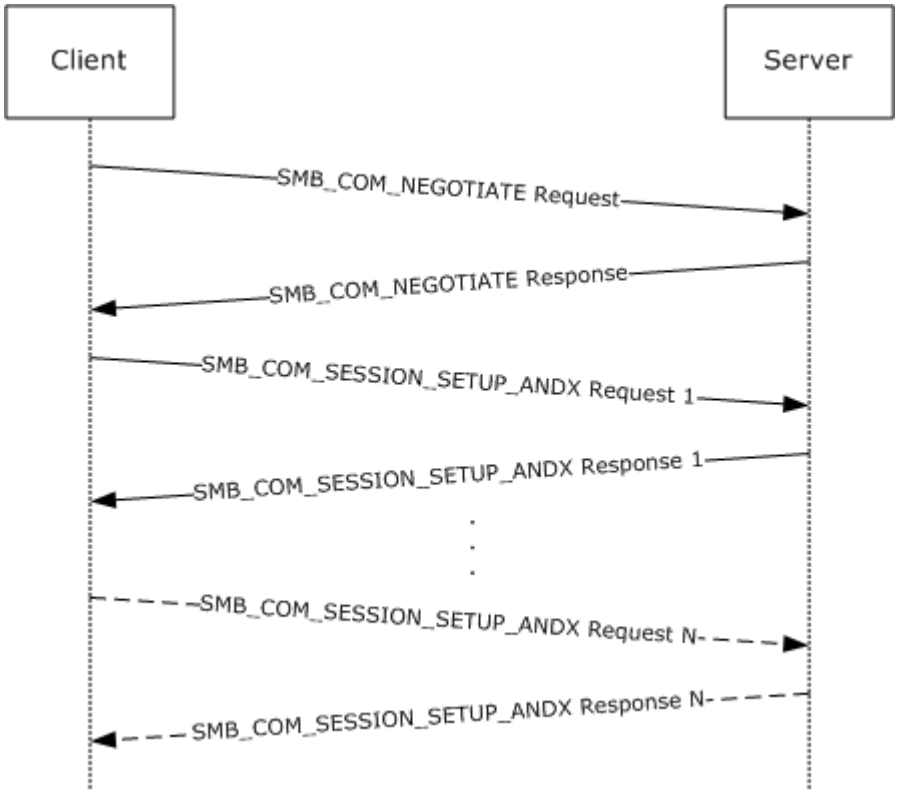


Figure 2: User authentication and session establishment sequence

The diagram illustrates the sequence of events during and after the SMB_COM_NEGOTIATE command is sent to the server, and the response is received.

Session Setup Round Trip

The SMB client examines two fields in the SMB_COM_NEGOTIATE server response to build the SMB_COM_SESSION_SETUP_ANDX request:

- The SMB client examines the **Capabilities** field in the SMB_COM_NEGOTIATE server response (section 2.2.3). If this field indicates that the SMB server supports extended security (that is, the CAP_EXTENDED_SECURITY bit is set), the SMB client MUST build an SMB_COM_SESSION_SETUP_ANDX request in the extended form. Otherwise, the SMB client MUST build an SMB_COM_SESSION_SETUP_ANDX request in the non-extended form.

- The SMB client examines the **DialectName** field in the SMB_COM_NEGOTIATE Server response. If it indicates that it does not respond to SMB dialect NTLM 0.12 (see section 2.2.2) but uses a down-level dialect, the SMB client MUST build a SMB_COM_SESSION_SETUP_ANDX (non-extended) request, and MUST set the **WordCount** field to 0x0a (10), as specified in [CIFS] section 4.1.2.

If the SMB server responds to SMB dialect NTLM 0.12, the following two cases exist:

- If the server supports extended security, the SMB client MUST build an extended SMB_COM_SESSION_SETUP_ANDX request, and MUST set the **WordCount** field to 0x0c (12). Otherwise, the SMB client MUST build an SMB_COM_SESSION_SETUP_ANDX request, and MUST set the **WordCount** field to 0x0d (13).

  The SMB client MUST check that the SMB server is configured with user level security or share level security (for information about the NEGOTIATE_USER_SECURITY bit in the **SecurityMode** field in the SMB_COM_NEGOTIATE server response, see section 2.2.3).

  If the SMB server is configured with share level security, the authentication is done using the SMB_COM_TREE_CONNECT_ANDX request as specified in [CIFS] sections 2.10 and 4.1.4.

  If the SMB server is configured with user level security, the SMB client MUST build the SMB_COM_SESSION_SETUP_ANDX request request as follows:

- If the SMB client decides to build a non-extended SMB_COM_SESSION_SETUP_ANDX request, it MUST check that:

  - The NEGOTIATE_ENCRYPT_PASSWORDS bit is set in the **SecurityMode** field of the SMB_COM_NEGOTIATE Server response (section 2.2.3).

  - If the SMB_COM_NEGOTIATE server response server has the bit NEGOTIATE_ENCRYPT_PASSWORDS set in **SecurityMode** field, the SMB client encrypts the password. If this bit is not set, the SMB client MAY copy the password in plain text.

  - If the SMB server supports UNICODE (for information on how the UC bit is set in the **Flags2** field in the SMB header, see section 2.2.1), the SMB client MUST copy the password as additional data at the end of the **CaseSensitivePassword** and **CaseSensitivePasswordLength** fields appropriately (for more information, see section 3.2.4.2.3).

  - If the SMB server does not support UNICODE, the SMB client MUST copy the encrypted password as additional data at the end of the **CaseInsensitivePasswordLength** and **CaseInsensitivePassword** fields. More information is specified in section 3.2.4.2.3.

In both cases, the password is copied as additional data at the end of the fixed fields.

If the SMB client decides to build an extended SMB_COM_SESSION_SETUP_ANDX request, it MUST ignore the NEGOTIATE_ENCRYPT_PASSWORDS bit set in the SecurityMode field of the SMB_COM_NEGOTIATE server response. Details are specified in section 3.2.4.2.3, [RFC4178] section 1.3.2, [MS-NLMP] (section), and [RFC2743].

If the client request was for an extended session setup and the server supports extended security, the SMB server builds an SMB_COM_SESSION_SETUP_ANDX server response (section 2.2.5). The security BLOB in the session setup response is built as specified in [RFC4178] section 1.3.2. If the client request does not indicate an extended session setup, the SMB server builds a response as specified in [CIFS] section 4.1.2.

Upon receiving the SMB_COM_SESSION_SETUP_ANDX server response (section 2.2.5), the SMB client invokes the local security package to determine if the session setup request SHOULD be continued, or if the session setup has been completed (that is, the server has enough information to establish the session) or aborted (that is, the server cannot proceed with the session setup because of an error in the information presented by the client, or otherwise).

If the session setup has to be continued, the security package on the client and/or server needs an additional round trip before the session setup can be established. This is especially true of new security packages that support mutual authentication between the client and server.

In the case of extended security, the SMB protocol does not make the distinction between NTLM and Kerberos (therefore, the sequence defined above is the same in both cases).

The protocol terminates after one round of session setup request and response is exchanged in the case of NTLM. And it MAY require multiple trips in the case of other security packages before the session setup can be established.

In the case of additional round trips, each round trip MUST consist of one SMB_COM_SESSION_SETUP_ANDX client request and one SMB_COM_SESSION_SETUP_ANDX server response. In the sequence diagram, this is represented in the vertical dotted line symbolizing additional round trips until the final round trip, represented as SMB_COM_SESSION_SETUP_ANDX Client Request N and SMB_COM_SESSION_SETUP_ANDX Server Response N, where N is a number larger than 1.

All additional SMB session setup round trips follow the same sequence details as Session Setup Round Trip, earlier in this topic.<148>

3.2.4.2.4 Tree Connect Establishment

If there is a tree connect already established to the target share in TreeConnectList, it SHOULD be reused. If not, the client creates an SMB_COM_TREE_CONNECT_ANDX request, as specified in [CIFS] section 4.1.4. These changes from tree connect establishment (as specified in [CIFS]) are defined below:<149>

- Signing Key Protection
  - Theclient MAY request signing key protection by setting the TREE_CONNECT_ANDX_EXTENDED_SIGNATURES flag in the OptionalSupport bits of the SMB_COM_TREE_CONNECT_ANDX request to TRUE.<150>
- Extended Information Response
  - Theclient MAY request extended information in the response, as defined in SMB_COM_TREE_CONNECT_ANDX Server Response Extension, by setting the TREE_CONNECT_ANDX_EXTENDED_RESPONSE flag in the OptionalSupport bits of the SMB_COM_TREE_CONNECT_ANDX request to TRUE.<151>

The client sends this message to the server.

3.2.4.3 Application Requests Opening a File

To open a file on a remote share, the application provides the path of the file being opened including the server and share that hosts the file, the security context of the user opening the file, the requested access for the open, the sharing mode for the open, the disposition action to take based on the existence of the target file, and other options the client MAY wish to pass in. These have not changed from what is specified in [CIFS] section 4.2.1. However, new to these extensions, the application could request that the maximal access information be returned on the open. The client MAY issue either an SMB_COM_NT_CREATE_ANDX operation or an SMB_COM_OPEN_ANDX operation. The latter is considered obsolete.<152>

- SMB_COM_NT_CREATE_ANDX Request
  - If the application has requested maximal access information, the client MUST request extended information in the response by setting the NT_CREATE_REQUEST_EXTENDED_RESPONSE flag in the **Flags** field of the SMB_COM_NT_CREATE_ANDX request (section 2.2.8) packet. The request also supports the use of the previous version token flag, as defined below. Other than this, the request is constructed as specified in [CIFS] section 4.2.1.

3.2.4.3.1 Scanning the Path for Previous Version Token

The client accesses a previous version of the file by placing a time indicator in the path as a directory element. For any path-based operation, including SMB_COM_NT_CREATE_ANDX, the client SHOULD scan the file path being sent in the request for an appropriate @GMT token. If the token is present in the path name as a directory element or final target, the client SHOULD set the SMB_FLAGS2_REPARSE_PATH flag in the SMB header of the SMB_COM_NT_CREATE_ANDX request (section 2.2.8):<153>

- SMB_COM_OPEN_ANDX Request
  - The client MAY request extended information in the response by setting the SMB_OPEN_EXTENDED_RESPONSE flag in the **Flags** field of the SMB_COM_OPEN_ANDX request packet.

    Because it is a path-based operation, this request follows the same Previous Version token parsing rules as specified in section 3.2.4.3.1. Other than this, the request is constructed as specified in [CIFS] section 5.8.<154>

3.2.4.4 Application Requests Reading from a File

The client requests a read from a file or named pipe by using the SMB_COM_READ_ANDX request (section 3.3.5.7). The application MUST provide the existing open on which to execute the read, the amount of data to be read, the offset into the file in bytes from where to read the data, and a buffer in which to place the data. This request is identical to what is specified in [CIFS] section 4.2.4, with the following exceptions:

- Reserved

  This field is modified to be a union of a 32-bit field named **Timeout** and a 16-bit number named **MaxCountHigh**. For file reads, if the read is larger than 0xFFFF, the client MUST use the **MaxCountHigh** field to hold the more significant two bytes of the requested size, allowing for a 32-bit read length. If the read is not larger than 0xFFFF, the client MUST set the **MaxCountHigh** to 0. For pipe reads, the client MAY set the **Timeout** field to a requested timeout value in seconds, or the client MAY set the **Timeout** field to 0xFFFFFFFF, indicating that the operation SHOULD NOT time out.

- Large Read Support

  If the client wants to issue a read for a size larger than MaxBufferSize, and the CAP_LARGE_READX bit is set in ServerCapabilities, the client MAY issue a read of a larger size. Otherwise, it MUST split the read into multiple requests to retrieve the entire amount of data. If a large read is being used, the length of the read MUST be placed in the MaxCount of the SMB_COM_READ_ANDX request. Other than this difference, the request is identical to what is specified in [CIFS] section 4.2.4.

The message is sent to the server.<155>

3.2.4.5 Application Requests Writing to a File

The client requests a write to a file or named pipe by using the SMB_COM_WRITE_ANDX request (section 3.3.5.7). The application MUST provide the existing open on which to execute the write, the amount of data being written, the offset into the file in bytes to where to write the data, and a buffer containing the data to write. This request is identical to what is specified in [CIFS] section 4.2.5, with the following exceptions:

- Remaining

  This field is ignored by the server in the case of writing to a file. This field is valid and relevant only for message mode pipe writes. If a pipe write spans over multiple requests, the client MUST set this field to the number of bytes remaining to be written.<156>

- WriteMode

  This field contains individual bit flags that indicate various details related to the write request. This field is modified to use additional bits, as follows.

0	1	2	3	4	5	6	7	8	9	10	1	2	3	4	5
X	X	X	X	MS	NP	X	DF	X	X	X	X	X	X	X	X

  X: Unused bit field. SHOULD be set to 0 when sent, and MUST be ignored when received.

  DF (SMB_WMODE_WRITE_THROUGH): If set, indicates that the write MUST be completed on disk and that the client expects the response to contain the status for the final result.

  NP (SMB_WMODE_WRITE_RAW_NAMED_PIPE): If set, indicates that this message request is to write data to a raw named pipe.

  MS (SMB_WMODE_START_OF_MESSAGE): This flag is only valid when SMB_WMODE_WRITE_RAW_NAMED_PIPE is set. If this flag is set, it indicates that the message contains the first part of a multipiece pipe write. The total length of the write MUST be given in the **Remaining** field, and the number of bytes contained in this request MUST be given in the **DataLength** field. Subsequent SMB_WMODE_WRITE_RAW_NAMED_PIPE write requests MUST be sent to write the remaining number of bytes.

- Large Write Support

  If the client wants to issue a write for a size larger than MaxBufferSize, and the CAP_LARGE_WRITEX bit is set in ServerCapabilities, the client MAY issue a write of a larger size. Otherwise, it MUST split the write into multiple requests to write the entire amount of data. Other than this difference, the request is identical to what is specified in [CIFS] section 4.2.4.

The message is sent to the server.<157>

3.2.4.6 Application Requests a Directory Enumeration

If the client requests an enumeration of a directory by issuing an SMB_COM_TRANSACTION2 request (section 2.2.13) with the TRANS2_FIND_FIRST2 (section 2.2.13.13) subcommand, as specified in [CIFS] section 4.3.4, the application MUST provide: the path to query information from, the amount of data to return, a wildcard qualifier that defines which files to return data for, the information level that defines the format of the data to return, any appropriate flags (as specified in [CIFS] section 4.3.4), and a buffer to receive the information returned. There are several new information levels supported in this extension as well as a wildcard form of the @GMT token:

- New Information Levels

  To request the new information levels, as specified in section 2.2.13.13, the client MUST set the InformationLevel of the TRANS2_FIND_FIRST2 request, as specified in [CIFS] section 4.3.3, to the corresponding InformationLevel in the table in section 2.2.13.13.

- Enumerating Previous Versions via FindFirst

  The client MAY attempt to enumerate the available snapshots (also known as previous versions) by issuing a TRANS2_FIND_FIRST2 request. The request MUST have the **FileName** field set to the search pattern to the @GMT token wildcard as a null-terminated array of 16-bit Unicode characters, and MUST request the enumeration at the SMB_FIND_FILE_BOTH_DIRECTORY_INFO InformationLevel. This request is not available for InformationLevels other than SMB_FIND_FILE_BOTH_DIRECTORY_INFO, and the client MAY fail such requests or simply pass-through the requests to the server.<158>

- Previous Version Parsing

  Because it is a path-based operation, this request follows the same Previous Version token parsing rules as specified in section 3.2.4.3.1.

The message is sent to the server. If the search finds no names that match the client request, the server returns STATUS_NO_MORE_FILES as a 32-bit error code if the client set SMB_FLAGS2_NT_STATUS in the **Flags2** field of the client request. Note that STATUS_NO_MORE_FILES is not considered an error in this case.

3.2.4.7 Application Requests Querying File Attributes

The client requests the retrieval of attributes from a file or named pipe by using the TRANS2_QUERY_FILE_INFORMATION (section 2.2.13.1) or TRANS2_QUERY_PATH_INFORMATION (section 2.2.13.3) subcommand of the SMB_COM_TRANSACTION2 request, as specified in [CIFS] sections 4.2.15 and 4.2.14. The client MAY use either command to query the attributes. The application MUST provide either the complete path to the file whose attributes are being queried or an existing open SMB Fid to the file being queried. It also MUST provide the information level that describes the format of information being queried and a buffer to receive the information returned. The SMB client MUST determine an appropriate information level to send to the SMB server to retrieve the requested file attributes. The extension adds support for pass-through of native-Windows NT attribute queries to the server side without translation. If the CAP_INFOLEVEL_PASSTHRU bit in ServerCapabilities is set, and the attribute query is for a native Windows NT information level, the client MAY execute a pass-through request, as defined below. Otherwise, it MUST choose an appropriate, existing SMB information level (as specified in [CIFS] section 4.2.14) and reconstruct the information required by the application. If an appropriate SMB information level is not available, the server MUST fail the request with STATUS_NOT_SUPPORTED.

An SMB server MAY not support streams on a specific share or on all shares. If the server does not support streams, it MUST set the NO_SUBSTREAMS bit in the **FileStatusFlags** field in the SMB_COM_NT_CREATE_ANDX server response (section 2.2.9) to an SMB_COM_NT_CREATE_ANDX client request (section 2.2.8).

A client MAY send a TRANS2_QUERY_FILE_INFORMATION subcommand of the SMB_COM_TRANSACTION2 request to the server with the **InformationLevel** field set to SMB_QUERY_FILE_STREAM_INFO. If the Fid field in the client request is on an SMB share that does not support streams, the server MUST fail the request with STATUS_NOT_SUPPORTED.

A client MAY send a TRANS2_QUERY_PATH_INFORMATION subcommand of the SMB_COM_TRANSACTION2 request to the server with the **InformationLevel** field set to SMB_QUERY_FILE_STREAM_INFO. If the **FileName** field in the client request is on an SMB share that does not support streams, the server MUST fail the request with STATUS_NOT_SUPPORTED.

Because the TRANS2_QUERY_PATH_INFORMATION subcommand request is a path-based operation, the path SHOULD be scanned for previous version tokens by the client, as specified in section 3.2.4.3.1. If an @GMT token is found, the SMB_FLAGS2_REPARSE_PATH bit flag MUST be set in the **Flags2** field of the SMB header of the client request.

Otherwise, the query takes the form as specified in [CIFS] sections 4.2.14 and 4.2.15:<159>

- Pass-Through Queries

  To execute a pass-through query, the client MUST increment the native information level by 1,000 (0x3e8). The resulting value is placed in the **InformationLevel** field in the TRANS2_QUERY_FILE_INFORMATION request. A list of supported information levels is specified in [MS-FSCC] section 2.4.

The message is sent to the server.

3.2.4.8 Application Requests Setting File Attributes

The client requests the setting of attributes on a file or named pipe by using the TRANS2_SET_FILE_INFORMATION (section 2.2.13.5) or TRANS2_SET_PATH_INFORMATION (section 2.2.13.7) subcommand of the SMB_COM_TRANSACTION2 (section 2.2.13) request, as specified in [CIFS] sections 4.2.17 and 4.2.16. The client MAY use either command to set the attributes. The application MUST provide either the path to the file whose attributes are being queried, or an existing SMB Fid to the file whose attributes are being altered. It also MUST provide the information level that describes the format of information being applied and a buffer containing the information. The SMB client MUST determine an appropriate information level to send to the SMB server to set the requested file attributes. The extension adds support for pass-through of native Windows NT attributes to the server side without translation. If the CAP_INFOLEVEL_PASSTHRU bit in ServerCapabilities is set, and the attribute being applied is a native Windows NT information level, the client MAY execute a pass-through request, as defined below.

Because the TRANS2_SET_PATH_INFORMATION request is a path-based operation, the path SHOULD be scanned by the client for previous version tokens, as specified in section 3.2.4.3.1. If an @GMT token is found, the SMB_FLAGS2_REPARSE_PATH bit MUST be set in the Flags2 field in the SMB header of the request.

Otherwise, the set operation takes the form as specified in [CIFS] sections 4.2.16 and 4.2.17:<160>

- Pass-Through Attribute Set

To execute a pass-through set, the client MUST increment the native information level by 1,000 (0x3e8). The resulting value is placed in the InformationLevel field in the TRANS2_QUERY_FILE_INFORMATION request. The serialized native structure is placed in the Data buffer of the SMB_COM_TRANSACTION2 request, and the TotalDataCount is set to the length of this buffer. A list of supported information levels is specified in [MS-FSCC] section 2.4.

The message is sent to the server.

3.2.4.9 Application Requests Querying File System Attributes

The client requests the retrieval of attributes from a file system by using the TRANS2_QUERY_FS_INFORMATION (section 2.2.13.9) subcommand of the SMB_COM_TRANSACTION2 (section 2.2.13) request, as specified in [CIFS] section 4.1.6. The application MUST provide an existing open to a file on the share whose file system is being queried. It also MUST provide the information level that describes the format of information being queried and a buffer to receive the information returned. The SMB client MUST determine an appropriate information level to send to the SMB server to retrieve the requested file system attributes. The extension adds support for pass-through of native Windows NT attribute queries to the server side without translation. If the CAP_INFOLEVEL_PASSTHRU bit in ServerCapabilities is set, and the attribute query is for a native Windows NT information level, the client MAY execute a pass-through request as described later in this topic.

Otherwise, the query takes the form as specified in [CIFS] section 4.1.6:<161>

- Pass-Through Queries

  To execute a pass-through query, the client MUST increment the native information level by 1,000 (0x3e8). The resulting value is placed in the InformationLevel field in the TRANS2_QUERY_FS_INFORMATION request. A list of supported information levels is specified in [MS-FSCC] section 2.4.

3.2.4.10 Application Requests Setting File System Attributes

The client requests the setting of attributes on a file system by using the TRANS2_SET_FS_INFORMATION subcommand of the SMB_COM_TRANSACTION2 (section 2.2.13) request. This request is not defined in [CIFS], so its format is provided in TRANS2_SET_FS_INFORMATION Request (section 2.2.13.11). The application MUST provide an existing open to a file on the share whose file system is being altered. It also MUST provide the information level that describes the format of information being applied and a buffer containing the information. The SMB client MUST determine an appropriate information level to send to the SMB server to set the requested file system attributes. If the CAP_INFOLEVEL_PASSTHRU bit in ServerCapabilities is set, and the attribute being applied is a native Windows NTinformation level, the client MUST execute a pass-through request as described below. If the bit is not set, the client MUST return a failure code to the application that is initiating the request:<162>

- Pass-Through Attribute Set

  To execute a pass-through set, the client MUST increment the native information level by 1,000 (0x3e8). The resulting value is placed in the InformationLevel field in the TRANS2_SET_FS_INFORMATION request. The serialized native structure is placed in the Data buffer of the SMB_COM_TRANSACTION2 request, and the TotalDataCount is set to the length of this buffer. A list of supported information levels is specified in [MS-FSCC] section 2.5.

3.2.4.11 Application Requests Executing an I/O Control

A client requests executing an I/O control on a remote file by sending an SMB_COM_NT_TRANSACTION (section 2.2.14) request with an NT_TRANS_IOCTL (section 3.3.5.17) subcommand. The client places the parameters into the fields, as specified in [CIFS] section 4.6.1. A list of common Windows NT native FSCTLs is specified in [MS-FSCC] section 2.3. There are three I/O control codes that are specific to the extension and described in the following sections.

3.2.4.11.1 Application Requests Enumerating Available Previous Versions

An application that wants to retrieve an enumeration of available previous version time stamps for a share issues the FSCTL_ENUMERATE_SNAPSHOTS IO control code, as specified in section NT_TRANSACT_IOCTL Client Request Extension (section 2.2.14.7). The request is sent to the server.<163>

3.2.4.11.2 Application Requests a Server-Side Data Copy

An outline follows of the steps taken for a server-side data copy of an entire file. Similar steps can be used to copy partial ranges within a file. The client issues a query for the copychunk resume key on the source file. Then it MUST pass this value into each iteration of the copychunk FSCTL that copies a given range on the server side. The client continues to issue these copies for subsequent data ranges until the entire file has been copied.

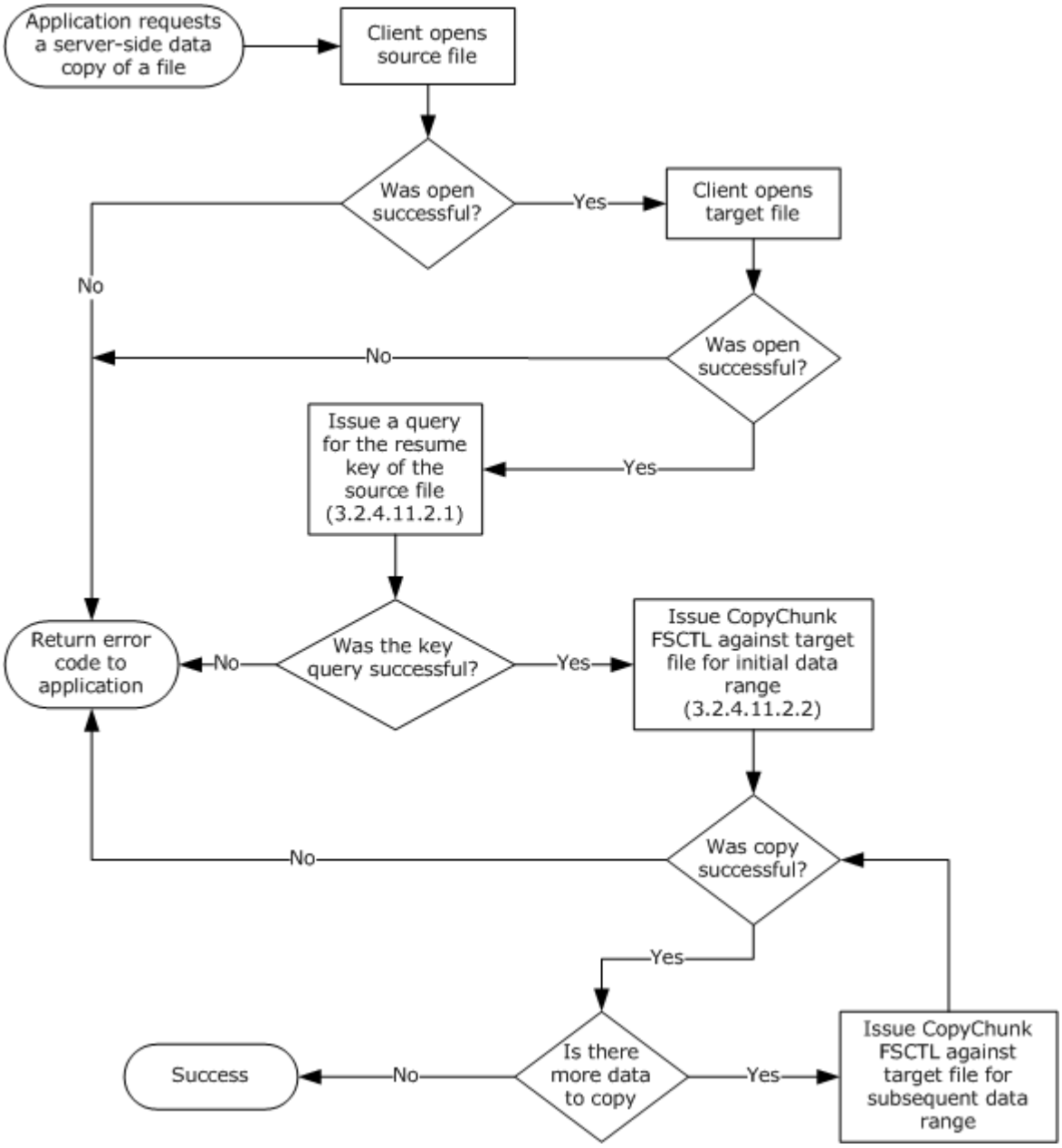


Figure 3: Server-side data copy of an entire file

3.2.4.11.2.1 Requesting the Copychunk Resume Key for a File

To request a copychunk resume key for an open file, the client issues an FSCTL_SRV_REQUEST_RESUME_KEY I/O control code, as specified in FSCTL_SRV_REQUEST_RESUME_KEY Request (section 2.2.14.7.2). The SMB_COM_NT_TRANSACTION request MUST be constructed as specified in FSCTL_SRV_REQUEST_RESUME_KEY Request and in [CIFS] section 3.13.2. The Fid of the source file is placed in the client request along with the FSCTL_SRV_REQUEST_RESUME_KEY function code. No parameter or data block is required. The request is sent to the server.

3.2.4.11.2.2 Requesting the Server-Side Copy of a Given Data Range

To request a server-side copy of a data range, the client MUST construct an SMB_COM_NT_TRANSACTION (section 2.2.14) request, as specified in FSCTL_SRV_COPYCHUNK Request (section 2.2.14.7.3). The copychunk resume key retrieved for the source file and the range of data blocks to be copied are placed in the Data buffer of the transaction request. The SMB file identifier of the destination file is placed in the Fid field of the request, and the function code is set to the FSCTL_SRV_COPYCHUNK value. This message is sent to the server.

3.2.4.12 Application Requests Querying DFS Referrals

To retrieve referral information, the DFS Referral application provides the SMB client with an opaque request buffer formatted as specified in [MS-DFSC] (section). for the payload of the SMB_COM_TRANSACTION2 (section 2.2.13) request. Upon receiving the buffer, the SMB client MUST issue an SMB_COM_TRANSACTION2 request with a TRANS2_GET_DFS_REFERRAL (section 2.2.13.17) subcommand, with the opaque request buffer as the parameter block of the TRANSACT2 request.

The DFS Referral application also provides a response buffer into which the SMB client copies the server response (for more information, see section 3.2.5.8).

For more information see [MS-DFSC].<164>

3.2.4.13 Application Requests Querying Quota Information

A client requests quota information by issuing an SMB_COM_NT_TRANSACTION request with the subcommand NT_TRANS_QUERY_QUOTA. The client MUST construct the request, as specified in NT_TRANSACT_QUERY_QUOTA Client Request Extension (section 2.2.14.3). The application provides an open to a file on a share whose volume is wanted to query quota information from. The application provides a buffer to receive the quota information and the maximum number of bytes to receive. The client can provide either a SID list or a start SID, as described below. The client MUST scope the information to be returned to match what the application has requested:

1. If the application provides a SID list, the client MUST include a list of security identifiers whose quota information is to be read. It does so by passing them in the SidList and setting the SidListLength to the length of the array. In this case, StartSidLength MUST be zero, and SidListLength MUST be non-zero.

2. If the application provides a start SID, the client MUST request that the enumeration be resumed by giving the last security identifier it has received in a previous enumeration. In this case, StartSidLength MUST be non-zero, and SidListLength MUST be zero.

3. If the application does not provide a SID list or a start SID, the client MUST request that the enumeration start at the beginning and return all that fit in the buffer. In this case, both StartSidLength and SidListLength MUST be zero.

4. If the application provides both a SID list and a start SID, the client MUST fail the request with STATUS_INVALID_PARAMETER.

The request is sent to the server.<165>

3.2.4.14 Application Requests Setting Quota Information

A client requests quota information by issuing an SMB_COM_NT_TRANSACTION request with the subcommand NT_TRANS_SET_QUOTA. The application provides an open to a file on a share whose volume is wanted to apply quota information to. The application provides the length of quota

information being applied and a buffer containing that information, as specified in NT_TRANSACT_QUERY_QUOTA Server Response Extension (section 2.2.14.4). The client MUST construct the request, as specified in NT_TRANSACT_SET_QUOTA Client Request Extension (section 2.2.14.5), and send the request to the server.

3.2.4.15 Application Requests Writing to a Mailslot

A client requests writing to a mailslot by issuing an SMB_COM_TRANSACTION request with the subcommand TRANS_MAILSLOT_WRITE. The application MUST provide the path to the mailslot that is being written to, the number of bytes to write, and a buffer containing the bytes to write. The client MUST construct the request, as specified in TRANS_MAILSLOT_WRITE Request (section 2.2.12.23), and send the request to the server.

3.2.4.16 Application Requests Setting Named Pipe State

A client requests setting the state of a named pipe by issuing an SMB_COM_TRANSACTION request with the subcommand TRANS_SET_NMPIPE_STATE. The application MUST provide an open to the pipe on which information is to be applied. The client MUST construct the request, as specified in section 2.2.12.1, and send the request to the server.

3.2.4.17 Application Requests Querying Named Pipe Information

A client requests querying pipe information on a named pipe by issuing an SMB_COM_TRANSACTION request with the subcommand TRANS_QUERY_NMPIPE_INFO. The application MUST provide an open to the pipe whose information is to be queried and a buffer to receive the returned information. The client MUST construct the request, as specified in section 2.2.12.7, and send the request to the server.

3.2.4.18 Application Requests Peeking at Named Pipe Data

A client requests peeking into pipe data on a named pipe by issuing an SMB_COM_TRANSACTION request with the subcommand TRANS_PEEK_NMPIPE. The application MUST provide an open to the pipe whose data is to be read, the number of bytes to attempt to read, and a buffer to receive the bytes read. The client MUST construct the request, as specified in section 2.2.12.9, and send the request to the server.

3.2.4.19 Application Requests Executing a Transaction on a Named Pipe

A client executes a transaction on a named pipe by issuing an SMB_COM_TRANSACTION (section 2.2.12) request with the subcommand TRANS_TRANSACT_NMPIPE. The application MUST provide an open to the pipe on which to perform the transaction, the number of bytes to write, a buffer of data to write into the pipe, a size of the maximum number of bytes to read out of the pipe, and a buffer to receive the data that is to be read. The client MUST construct the request, as specified in section 2.2.12.11, and send the request to the server.

3.2.4.20 Application Requests Waiting for Named Pipe Availability

A client requests waiting for named pipe availability by issuing an SMB_COM_TRANSACTION (section 2.2.12) request with the subcommand TRANS_WAIT_NMPIPE. The application MUST provide the name of the pipe that is to be waited on. The client MUST construct the request, as specified in section 2.2.12.19, and send the request to the server.

3.2.4.21 Application Requests Querying Named Pipe Handle State

A client queries named pipe state by issuing an SMB_COM_TRANSACTION (section 2.2.12) request with the subcommand TRANS_QUERY_NMPIPE_STATE. The application MUST provide an open to the pipe whose state is being queried and a buffer in which to place the returned information. The client MUST construct the request, as specified in section 2.2.12.5, and send the request to the server.

3.2.4.22 Application Requests the Session Key for a Connection

An application MUST provide the security context for the user whose session key it wants to query and an indicator of the connection it wants to query the key for. This indicator can be either the name of a server that was connected to it, or an open file on an existing connection that can be used to locate the connection. The SMB client MUST locate the appropriate SMB connection and look up the session state in the SessionList based on the security context of the user. If no entry is found, the application request MUST be failed with STATUS_OBJECT_NAME_NOT_FOUND. If a session state entry is found, but the session key state is Unavailable, the request MUST be failed with STATUS_ACCESS_DENIED. Otherwise, the 16-byte session key for this session MUST be returned to the calling application.

3.2.4.23 Application Requests a Named RAP Transaction

An application can request a SMB_COM_TRANSACTION with the following parameters:

1. A transaction name. The SMB client MUST use this name as the transaction name in the SMB_COM_TRANSACT request.

2. A request "parameter" buffer and its length in bytes. The SMB client MUST send the contents of this buffer to the server in the parameter block of the SMB_COM_TRANSACTION request. The **TotalParamterCount** in the transact request MUST be set to the length of the buffer.

3. A response "parameter" buffer and its length in bytes. The SMB client MUST send the length to the server via the **MaxParameterCount** field of the SMB_COM_TRANSACTION request. The parameter block returned by the server MUST be copied to the buffer supplied by the client and the length in bytes of the response parameter buffer MUST be returned to the client.

4. A request "data" buffer and its length in bytes. The SMB client MUST send the contents of this buffer to the server in the data block of the SMB_COM_TRANSACTION request. The **TotalDataCount** in the transact request MUST be set to the length of the buffer.

5. A response "data" buffer and its length in bytes. The SMB client MUST send the length to the server via the **MaxDataCount** field of the SMB_COM_TRANSACTION request. The data block returned by the server MUST be copied to the buffer supplied by the client and the length in bytes of the response data buffer MUST be returned to the client.

6. The security context for the user requesting the operation.

7. The name of the SMB server to issue the request.

If an authenticated tree connection (for the user specified in (6)) to the IPC$ share on the server (specified in (7)) does not exist, the SMB client MUST connect to the IPC$ share as specified in section 3.2.4.2. The SMB client MAY use an existing tree connection/session for the user if one already exists. The UID for the session and the TID for the IPC$ share MUST be copied to the corresponding fields in the SMB header.

The **SetupCount** and the **MaxSetupCount** fields in the transact request MUST be set to 0. The format of an SMB_COM_TRANSACTION request is specified in [CIFS] section 3.13.1.

3.2.5 Message Processing Events and Sequencing Rules

3.2.5.1 Receiving Any Message

- Signing

  If a message is received and IsSigningActive is TRUE for the connection, the signature MUST be verified, as specified in section 3.1.5.1, unless the message is an oplock break, as specified in [CIFS] section 2.7.

  The client is responsible for providing the expected sequence number for signature verification. The sequence number for the incoming response is determined by what was stored in the ClientResponseSequenceNumber table. The client MUST look up the expected sequence number in that table based on the Mid of the response. The client uses ClientResponseSequenceNumber[Mid] as the sequence number in signature verification, as specified in section 3.1.5.1. If signature verification fails, the message MUST be discarded and not processed. The client MAY choose to disconnect the underlying connection and tear down all state associated with this connection.<166>

- Session Expiration

  If the request passed a valid authenticated session identifier in the **Uid** field in the SMB header, and the Status code in the SMB header of the response is STATUS_NETWORK_SESSION_EXPIRED, the client MUST attempt to reauthenticate this session. The client MUST look up the session state for this Uid in SessionList, and set the state to Expired. Then it follows the steps as specified in section 3.2.4.2.3, except that the Uid sent in the SMB header of the SMB_COM_SESSION_SETUP_ANDX request MUST be set to the **Uid** value taken from the session state. If the authentication fails, the resulting error code MUST be returned for whichever operation failed with STATUS_NETWORK_SESSION_EXPIRED; and the session state for this Uid is removed from SessionList. If authentication succeeds, the client MUST set the session state for this Uid in SessionList to Valid and retry the operation that failed with STATUS_NETWORK_SESSION_EXPIRED.

3.2.5.2 Receiving an SMB_COM_NEGOTIATE Response

Processing of an SMB_COM_NEGOTIATE response is handled as specified in [CIFS] with the following additions or clarifications:

- Determining server signing mode

  The server response indicates if the server has message signing enabled or required. If the NEGOTIATE_SECURITY_SIGNATURES_REQUIRED bit in the **SecurityMode** field of the SMB_COM_NEGOTIATE response is set, the client MUST set ServerSigningState to Required and ignore the NEGOTIATE_SECURITY_SIGNATURES_ENABLED bit. Otherwise, if the NEGOTIATE_SECURITY_SIGNATURES_ENABLED bit in the **SecurityMode** field of the SMB_COM_NEGOTIATE response is set, the client MUST set ServerSigningState to Enabled. If neither bit is set, the client MUST set ServerSigningState to Disabled.

  Once ServerSigningState is set, the client can consult the table in section 3.2.4.2.3 to determine if connectivity is blocked. If so, the connection SHOULD be terminated by disconnecting the underlying transport and tearing down any state associated with the connection.

- Storing server capabilities

  The client MUST store the capabilities returned in the SMB_COM_NEGOTIATE response in ServerCapabilities.

- Storing extended security token

  If the capabilities returned in the SMB_COM_NEGOTIATE response include CAP_EXTENDED_SECURITY, the response MUST take the form defined in section 2.2.3, and the client MUST set the GSSNegotiateToken to the value returned in the **SecurityBlob** field in the SMB_COM_NEGOTIATE server response.

- Storing NTLM encryption key

  If the capabilities returned in the SMB_COM_NEGOTIATE response do not include CAP_EXTENDED_SECURITY, the response MUST take the form as specified in [CIFS] section 4.1.1 for the NTLM 0.12 dialect, and the client MUST set the NTLMEncryptionKey to the value returned in the EncryptionKey of the SMB_COM_NEGOTIATE server response.

- Storing negotiate maximum buffer size

  The client MUST set **MaxBufferSize** to the value received in the **MaxBufferSize** field of the negotiate response.

  If negotiate is being processed as part of a connect attempt, the client continues to user authentication, as specified in section 3.2.4.2.3.

3.2.5.3 Receiving an SMB_COM_SESSION_SETUP_ANDX Response

Processing of an SMB_COM_SESSION_SETUP_ANDX response is handled as specified in [CIFS] sections 2.10 and 4.1.2 with the following additions and clarifications.

- Extended Security Authentication

  If the ServerCapabilities have the CAP_EXTENDED_SECURITY bit set, the client MUST reject any SMB_COM_SESSION_SETUP_ANDX responses that do not take the form specified in section 2.2.5. If the **Status** field of the SMB header is not STATUS_SUCCESS and is not STATUS_MORE_PROCESSING_REQUIRED, the authentication has failed, and the error code MUST be propagated back to the application that initiated the connection attempt. Otherwise, if there is no entry in SessionList for the Uid in the response, one MUST be created, and the state MUST be set to InProgress and associated with the security context of the user that initiated the authentication attempt. The session key for this session entry is set to an array of 16 zeros, and the session key state is set to Unavailable.

  The client MUST process the GSS token (the SecurityBlob field of the response with its length given in the SecurityBlobLength field). If this token is of nonzero length, the client MUST use the configured GSS authentication protocol to obtain the next GSS token for the authentication exchange. Based on the status code received in the response and the result from the GSS authentication protocol, one of the following actions MUST be taken.

  - If the GSS authentication protocol indicates an error, the error MUST be returned to the calling application that initiated the connection; and if session state was associated with this authentication, it MUST be removed from the SessionList.

  - If the **Status** field of the response contains STATUS_SUCCESS and the GSS authentication protocol does not indicate an error, authentication is complete. The session state for this Uid in the SessionList MUST be set to Valid.

  - If the **Status** field of the response contains STATUS_MORE_PROCESSING_REQUIRED and the GSS authentication protocol did not indicate an error, the client MUST create an SMB_COM_SESSION_SETUP_ANDX (section 2.2.4) packet with the following parameters. The client MUST set CAP_EXTENDED_SECURITY in the **Capabilities** field and set

SMB_FLAGS2_EXTENDED_SECURITY in the SMB header **Flags2** field. The **SecurityBlob** and **SecurityBlobLength** fields MUST contain the output token generated by the GSS protocol executed above. The **Uid** field MUST be set to the session identifier returned in the SMB header of the previous SMB_COM_SESSION_SETUP_ANDX response. This packet MUST be sent to the server, and further processing listed in the remainder of this section is not necessary.

- NTLM Authentication

  If the CAP_EXTENDED_SECURITY bit in ServerCapabilities is not set, the client processes the response as described here. If the **Status** field of the response does not contain STATUS_SUCCESS, the client MUST propagate the error to the application that initiated the authentication. The connection MUST remain open for the client to attempt another authentication.

  If the **Status** field of the response contains STATUS_SUCCESS, authentication was successful. The client associates the returned Uid from the SMB header of the response with this user for further requests, as specified in [CIFS]. The session state for this Uid in SessionList MUST be set to Valid. If the session key associated with this session is all zeros, the client MUST query the authentication package for the 16-byte session key, as specified in [MS-NLMP], and set it in the session state. If the session key is nonzero, the client MUST NOT overwrite the existing session key.

- Activating Signing

  If authentication has just completed successfully, IsSigningActive is FALSE, and the targeted behavior for this connection is Signed based on the description in section 3.2.4.2.3, the client MUST determine if signing needs to be activated.

  This is done by determining the user context that completed authentication. If the user who authenticated is a guest (indicated by bit 0 of the **Action** field in the SMB_COM_SESSION_SETUP_ANDX response being set) or is anonymous (did not provide credentials), signing MUST NOT be activated.

  If neither of these conditions is true, the client MUST activate signing. It MUST query the authentication protocol (either NTLM or via GSS-API) for the session key used in this authentication and store it as ConnectionSigningSessionKey. If CAP_EXTENDED_SECURITY is set in ServerCapabilities, it MUST set ConnectionSigningUseNTLMChallengeResponse to FALSE. If CAP_EXTENDED_SECURITY is not set, it MUST set ConnectionSigningUseNTLMChallengeResponse to TRUE, and set ConnectionSigningNTLMChallengeResponse to the challenge response sent in the SMB_COM_SESSION_SETUP_ANDX response.

  Once these steps are done, the client MUST verify the signature of this response. The client follows the steps specified in section 3.1.5.1, passing in a sequence number of 1 because this is the first signed packet.

3.2.5.4 Receiving an SMB_COM_TREE_CONNECT_ANDX Response

Processing of an SMB_COM_TREE_CONNECT_ANDX response is handled and the returned Tid is stored, as specified in [CIFS] section 3.1.3, with the following additions. The tree connect entry is created in TreeConnectList using the Tid of the response and the share name that was sent in the request. **MaximalAccess** and **GuestMaximalAccess** are set to 0.

- Receiving Extended Information

The client MUST determine if the server returned an extended response, as specified in section 2.2.7. It does so by determining if the **WordCount** is equal to 7. If it is, the client MUST make the new extended information available to the calling application by setting the **MaximalAccess** and **GuestMaximalAccess** values for the tree connect in the TreeConnectList for this Tid to the values received from the server.

- Negotiating Signing Key Protection

  If the response status is STATUS_SUCCESS, and the SMB_EXTENDED_SIGNATURE bit is set in the **OptionalSupport** field of the SMB_COM_TREE_CONNECT_ANDX response, the client MUST hash the session key of the calling user. This protects the key used for signing by making it unavailable to calling applications.

  The one-way hash done on the user session key uses the HMAC-MD5 algorithm, as specified in [RFC2104]. The steps are as follows:

  1. Take the 16-byte user session key. If this is an LM authentication where the session key is only 8 bytes, zero extend it to 16 bytes.

  2. Calculate the one-way hash as follows. The resulting 16-bit digest is treated as the user's new session key and returned to any appropriate caller who requests it. SSKeyHash is the well-known constant array that is described in the following example.

```
CALL hmac_md5( SSKeyHash, 256, session key, session key length, digest )
SET user session key = digest
BYTE SSKeyHash[256] = {
0x53, 0x65, 0x63, 0x75, 0x72, 0x69, 0x74, 0x79,
0x20, 0x53, 0x69, 0x67, 0x6e, 0x61, 0x74, 0x75,
0x72, 0x65, 0x20, 0x4b, 0x65, 0x79, 0x20, 0x55,
0x70, 0x67, 0x72, 0x61, 0x64, 0x65, 0x79, 0x07,
0x6e, 0x28, 0x2e, 0x69, 0x88, 0x10, 0xb3, 0xdb,
0x01, 0x55, 0x72, 0xfb, 0x74, 0x14, 0xfb, 0xc4,
0xc5, 0xaf, 0x3b, 0x41, 0x65, 0x32, 0x17, 0xba,
0xa3, 0x29, 0x08, 0xc1, 0xde, 0x16, 0x61, 0x7e,
0x66, 0x98, 0xa4, 0x0b, 0xfe, 0x06, 0x83, 0x53,
0x4d, 0x05, 0xdf, 0x6d, 0xa7, 0x51, 0x10, 0x73,
0xc5, 0x50, 0xdc, 0x5e, 0xf8, 0x21, 0x46, 0xaa,
0x96, 0x14, 0x33, 0xd7, 0x52, 0xeb, 0xaf, 0x1f,
0xbf, 0x36, 0x6c, 0xfc, 0xb7, 0x1d, 0x21, 0x19,
0x81, 0xd0, 0x6b, 0xfa, 0x77, 0xad, 0xbe, 0x18,
0x78, 0xcf, 0x10, 0xbd, 0xd8, 0x78, 0xf7, 0xd3,
0xc6, 0xdf, 0x43, 0x32, 0x19, 0xd3, 0x9b, 0xa8,
0x4d, 0x9e, 0xaa, 0x41, 0xaf, 0xcb, 0xc6, 0xb9,
0x34, 0xe7, 0x48, 0x25, 0xd4, 0x88, 0xc4, 0x51,
0x60, 0x38, 0xd9, 0x62, 0xe8, 0x8d, 0x5b, 0x83,
0x92, 0x7f, 0xb5, 0x0e, 0x1c, 0x2d, 0x06, 0x91,
0xc3, 0x75, 0xb3, 0xcc, 0xf8, 0xf7, 0x92, 0x91,
0x0b, 0x3d, 0xa1, 0x10, 0x5b, 0xd5, 0x0f, 0xa8,
0x3f, 0x5d, 0x13, 0x83, 0x0a, 0x6b, 0x72, 0x93,
0x14, 0x59, 0xd5, 0xab, 0xde, 0x26, 0x15, 0x6d,
0x60, 0x67, 0x71, 0x06, 0x6e, 0x3d, 0x0d, 0xa7,
0xcb, 0x70, 0xe9, 0x08, 0x5c, 0x99, 0xfa, 0x0a,
0x5f, 0x3d, 0x44, 0xa3, 0x8b, 0xc0, 0x8d, 0xda,
0xe2, 0x68, 0xd0, 0x0d, 0xcd, 0x7f, 0x3d, 0xf8,
0x73, 0x7e, 0x35, 0x7f, 0x07, 0x02, 0x0a, 0xb5,
0xe9, 0xb7, 0x87, 0xfb, 0xa1, 0xbf, 0xcb, 0x32,
0x31, 0x66, 0x09, 0x48, 0x88, 0xcc, 0x18, 0xa3,
0xb2, 0x1f, 0x1f, 0x1b, 0x90, 0x4e, 0xd7, 0xe1
};
```

After the session key has been hashed, or if the SMB_EXTENDED_SIGNATURE bit is not set, the client MUST start allowing applications to query the session key by setting the session key state on the session entry in the SessionList (based on this Uid) to Available.

3.2.5.5 Receiving an SMB_COM_NT_CREATE_ANDX Response

The client MUST determine whether the server returned an extended response, as specified in section 2.2.9. It does so by checking if the **WordCount** is equal to 42. If the response is not an extended response, the client MUST process the response, as specified in [CIFS] section 4.2.1. If the response is an extended response, the new information specified in section 2.2.9 MUST be propagated back to the calling application by completing the call initiated as specified in section 3.2.4.3 and returning the extended information.

3.2.5.6 Receiving an SMB_COM_OPEN_ANDX Response

The client MUST determine whether the server returned an extended response, as specified in section 2.2.11. It does so by checking if the **WordCount** is equal to 19. If the response is not an extended response, the client MUST process the response, as specified in [CIFS] section 5.8. If the response is an extended response, the new information specified in section 2.2.10 MUST be propagated back to the calling application by completing the call initiated as specified in section 3.2.4.3 and returning the extended information.

3.2.5.7 Receiving an SMB_COM_READ_ANDX Response

Other than the changes listed below, the response is processed as specified in [CIFS] section 4.2.4 with the following clarifications:

- DataCompactionMode: A 16-bit number that is unused and MUST be ignored.
- Pad: Optional padding to set the **Data** field to start on a 4-byte boundary from the start of the SMB header.
- The first two bytes of the **Reserved[5]** field, as specified in [CIFS], are used as the 16-bit **DataCountHigh** field. This field MUST contain the high 16-bits of a 32-bit length if the number of bytes is equal to or greater than 64 KB. Otherwise, this field MUST be set to 0. The remaining 3 bytes of **Reserved** remain unused and MUST be set to 0.

If the response indicates success, the client MUST return the data received to the application that initiated the call.

If the response indicates an error occurred, the client MUST propagate the error to the application that initiated the call. If the operation was a read on a named pipe, and the error code returned is STATUS_BUFFER_OVERFLOW, both the error code and the data MUST be returned to the calling application. This indicates there is more data remaining in the current message. The application is responsible for issuing a subsequent read operation to retrieve the remainder of the message.

3.2.5.8 Receiving an SMB_COM_WRITE_ANDX Response

The client MUST propagate the success or failure code from the response to the application that initiated the call. For the success case, the number of bytes written to the file MUST also be propagated.

3.2.5.9 Receiving a TRANS2_FIND_FIRST2 Response

If the response indicates an error occurred, the client MUST propagate the error to the application that initiated the call.

If the response indicates the operation is successful, the client MUST return the information received in the Data block of the response to the application that initiated the call. The client SHOULD validate that the data buffer is well formed; and, if any discrepancy is found, it SHOULD return a failure code to the calling application instead of propagating a badly formed data response from the server.

3.2.5.10 Receiving a TRANS2_FIND_NEXT2 Response

If the response indicates an error occurred, the client MUST propagate the error to the application that initiated the call.

If the response indicates the operation is successful, the client MUST return the information received in the Data block of the response to the application that initiated the call. The client SHOULD validate that the data buffer is well formed; and, if any discrepancy is found, it SHOULD return a failure code to the calling application instead of propagating a badly formed data response from the server.

3.2.5.11 Receiving a TRANS2_QUERY_FILE_INFORMATION Response

If the response indicates an error occurred, the client MUST propagate the error to the application that initiated the call.

If the response indicates the operation is successful, the client MUST return the information received in the Data block of the response to the application that initiated the call. The client SHOULD validate that the data buffer is well formed; and, if any discrepancy is found, it SHOULD return a failure code to the calling application instead of propagating a badly formed data response from the server.

3.2.5.12 Receiving a TRANS2_QUERY_PATH_INFORMATION Response

If the response indicates an error occurred, the client MUST propagate the error to the application that initiated the call.

If the response indicates the operation is successful, the client MUST return the information received in the Data block of the response to the application that initiated the call. The client SHOULD validate that the data buffer is well formed; and, if any discrepancy is found, it SHOULD return a failure code to the calling application instead of propagating a badly formed data response from the server.

3.2.5.13 Receiving a TRANS2_SET_FILE_INFORMATION Response

The client MUST propagate the success or failure code in the response to the application that initiated the call.

3.2.5.14 Receiving a TRANS2_SET_PATH_INFORMATION Response

The client MUST propagate the success or failure code in the response to the application that initiated the call.

3.2.5.15 Receiving a TRANS2_QUERY_FS_INFORMATION Response

If the response indicates an error occurred, the client MUST propagate the error to the application that initiated the call.

If the response indicates the operation is successful, the client MUST return the information received in the Data block of the response to the application that initiated the call. The client SHOULD validate that the data buffer is well formed; and, if any discrepancy is found, it SHOULD return a failure code to the calling application instead of propagating a badly formed data response from the server.

3.2.5.16 Receiving a TRANS2_SET_FS_INFORMATION Response

The client MUST propagate the success or failure code in the response to the application that initiated the call.

3.2.5.17 Receiving an NT_TRANS_IOCTL Response

If the response indicates an error occurred, the client MUST propagate the error to the application that initiated the call.

If the response indicates the operation is successful, the client MUST return the information received in the Data block of the response to the application that initiated the call.

3.2.5.17.1 Receiving an FSCTL_ENUMERATE_SNAPSHOTS Response

If the response indicates an error occurred, the client MUST propagate the error to the application that initiated the call.

If the response indicates the operation is successful, the client MUST return the information received in the Data block of the response to the application that initiated the call.

3.2.5.17.2 Receiving an FSCTL_SRV_REQUEST_RESUME_KEY Response

If the response indicates an error occurred, the client MUST propagate the error to the application that initiated the call.

If the response indicates the operation is successful, the client MUST return the copychunk resume key received in the Data block of the response to the application that initiated the call. If this is a server-side data copy operation, execution MUST continue as specified in section 3.2.4.11.2.

3.2.5.17.3 Receiving an FSCTL_SRV_COPYCHUNK Response

The success or failure code MUST be returned to the calling application. Also, the FSCTL_SRV_COPYCHUNK response information, as specified in FSCTL_SRV_COPYCHUNK response (section 2.2.14.8.3), MUST also be returned to the calling application in both success and failure situations. If this is a server-side data copy operation, execution MUST continue, as specified in section 3.2.4.11.2.2.

3.2.5.18 Receiving a TRANS2_ GET_ DFS_REFERRAL Response

If the response indicates an error occurred, the client MUST propagate the error to the application that initiated the call.

If the response indicates the operation is successful, the client MUST return the information received in the Data block of the TRANSACT2 response to the application that initiated the call.

3.2.5.19 Receiving a TRANS2_QUERY_QUOTA_INFO Response

If the response indicates an error occurred, the client MUST propagate the error to the application that initiated the call.

If the response indicates the operation is successful, the client MUST return the information received in the Data block of the response to the application that initiated the call.

3.2.5.20 Receiving a TRANS2_SET_QUOTA_INFO Response

The client MUST propagate the success or failure code in the response to the application that initiated the call.

3.2.5.21 Receiving a TRANS2_MAILSLOT_WRITE Response

The client MUST propagate the success or failure code in the response to the application that initiated the call.

3.2.5.22 Receiving a TRANS2_SET_NMPIPE_STATE Response

The client MUST propagate the success or failure code in the response to the application that initiated the call.

3.2.5.23 Receiving a TRANS2_QUERY_NMPIPE_INFO Response

If the response indicates an error occurred, the client MUST propagate the error to the application that initiated the call.

If the response indicates the operation is successful, the client MUST return the information received in the Data block of the response to the application that initiated the call.

3.2.5.24 Receiving a TRANS2_PEEK_NMPIPE Response

If the response indicates the operation is successful, the client MUST return the data received in the Data block of the response to the application that initiated the call.

If the response indicates an error occurred, the client MUST propagate the error to the application that initiated the call.

3.2.5.25 Receiving a TRANS2_TRANSACT_NMPIPE Response

If the response indicates the operation is successful, the client MUST return the data received in the Data block of the response to the application that initiated the call.

If the response indicates an error occurred, the client MUST propagate the error to the application that initiated the call. If the error returned is STATUS_BUFFER_OVERFLOW, this indicates that there is still more data to be read for the response message. In this case, the client MUST propagate both the error and the data received in the Data buffer of the response to the application that initiated the call. The application is responsible for issuing a subsequent read operation to retrieve the remainder of the message.

3.2.5.26 Receiving a TRANS2_WAIT_NMPIPE Response

If the response indicates an error occurred, the client MUST propagate the error to the application that initiated the call.

If the response indicates the operation is successful, the client MUST return the success to the application that initiated the call to indicate that the pipe is available.

3.2.5.27 Receiving a TRANS_QUERY_NMPIPE_STATUS Response

If the response indicates an error occurred, the client MUST propagate the error to the application that initiated the call.

If the response indicates the operation is successful, the client MUST return the information received in the Parameters block of the response to the application that initiated the call.

3.2.6 Timer Events

There are no new timers other than what is required by the CIFS Protocol, as specified in [CIFS].

3.2.7 Other Local Events

There are no new local events other than what is required by the CIFS Protocol, as specified in [CIFS].

3.3 Server Details

3.3.1 Abstract Data Model

No additional data is required beyond what is required by the CIFS Protocol, as specified in [CIFS], except for what is included in section 3.1.1 and the following additional parameters.

3.3.1.1 Global

There is no new global state. The CIFS Protocol, as specified in [CIFS], implies that the server holds a list of shares present on the machine. The list is as follows:

ShareList: A list of shares that are present on this server indexed by the share name.

GlobalCapabilities: Global capability bits supported by this server.

3.3.1.2 Per SMB Share

ShareName: The name of the share.

OptionalSupport: The optional support bits for the share.

ShareType: The type of share (that is, disk, printer, named pipe).

ShareSecurityDescriptor: The security descriptor that describes what users can access resources on that share and their corresponding access levels.

3.3.1.3 Per SMB Connection

ClientCapabilities: Capability flags of the client, as specified in section 2.2.3.

ServerNextReceiveSequenceNumber: Sequence number for the next signed request being received.

ServerSendSequenceNumber: A list of the expected sequence numbers for the responses of outstanding signed requests, indexed by message ID (Mid).

NTLMEncryptionKey: The encryption key received, sent by the server during a non-extended security negotiate for use in implicit NTLM authentication.

AuthenticationState: A table of authenticated sessions established on this connection via SMB_COM_SESSION_SETUP_ANDX requests that indicate, for a given Uid, if authentication is Expired, InProgress, or Complete.

SessionKeyEntry: A table that stores the session key and session key state for authenticated sessions to this server. The session key is a 16-byte value, and the session key state is either Available or Unavailable. The table can be searched by either the Uid of the session or by the user context of that session.

3.3.2 Timers

The Authentication Expiration Timer, a pre-authentication timer, is used to mark an authentication as expired when its authentication-specific expiration time is reached. NTLM authentication has no expiration time, so authentications done with NTLM do not expire. The default expiration for Kerberos is implementation specific.<167>

3.3.3 Initialization

When an SMB connection is established, the following values MUST be initialized.

- ServerNextReceiveSequenceNumber is set to 2.
- ServerSendSequenceNumber is set to an empty table.
- NTLMEncryptionKey is set to invalid.
- ClientCapabilities is set to 0.
- AuthenticationState is set to an empty table.
- ShareList is populated from system configuration to contain the list and properties of shares on this server.
- SessionKeyEntry is set to an empty table.

3.3.4 Higher-Layer Triggered Events

3.3.4.1 Sending Any Message

If a message is being sent and signing is active for the SMB connection, the message MUST be signed, as specified in section 3.1.4.1. As defined in that section, the logic below MUST be used for all sends once IsSigningActive has been set to TRUE except when the packet being sent is an oplock break, as specified in [CIFS] section 2.7.

The server is responsible for providing the appropriate sequence number for signature calculation. The sequence number for the next request is stored in the table ServerSendSequenceNumber based on the Mid of the request. (Mid is as specified in [CIFS] section 3.1.5.) The sequence number is

calculated and populated into the table, as specified in section 3.3.5.1. The sequence number ServerSendSequenceNumber[Mid] is used as specified in section 3.1.4.1 to sign the outgoing message.

If signing is not active, the **SecuritySignature** field of the SMB Header for all messages sent except SMB_COM_SESSION_SETUP_ANDX response MUST be set to zero. For SMB_COM_SESSION_SETUP_ANDX response, the **SecuritySignature** field of the SMB Header MUST be set to the one received in the SMB_COM_SESSION_SETUP_ANDX request.

3.3.4.2 Server Application Queries the Session Key

An application running on the server issues a query for a user session key by providing the context of the user whose session key they want. This can be done in several ways, one of which is to provide the open that the user has done to their named pipe and have the server do a reverse lookup to locate the session that opened that instance of the pipe. The application also provides a 16-byte buffer to receive the session key.

The server finds the session key state by locating the corresponding entry in the SessionKeyEntry table. If the entry is not found, the server MUST fail the request with STATUS_INVALID_PARAMETER. If the entry is found, but the state is Unavailable, the server MUST fail the request with STATUS_ACCESS_DENIED. Otherwise, the server with return the 16-byte session key for this user to the application.

3.3.4.3 DFS Server Notifies that It Is Active

In response to this event, the SMB server MUST set the CAP_DFS bit in the **GlobalCapabilities** as specified in section 3.3.1.1. If the DFS server is running on this computer, it MUST notify the SMB server that the DFS capability is available via this event.

3.3.4.4 DFS Server Notifies that a Share Is a DFS Share

In response to this event, the SMB server MUST set the SMB_SHARE_IS_IN_DFS bit in the **OptionalSupport** attribute of the share specified in section 3.3.1.2. If a DFS server running on this computer wants to claim a share as a DFS share, it MUST notify the SMB server via this event.

3.3.4.5 DFS Server Notifies that a Share Is Not a DFS Share

In response to this event, the SMB server MUST clear the SMB_SHARE_IS_IN_DFS bit in the **OptionalSupport** attribute of the share specified in section 3.3.1.2.

3.3.4.6 RAP Server Notifies that It Is Active

This event notifies the SMB server that a RAP server is available on the machine to process RAP requests, as specified in [MS-RAP].

3.3.5 Message Processing Events and Sequencing Rules

3.3.5.1 Receiving Any Message

- Signing

  If a message is received and IsSigningActive is TRUE for the SMB connection, the signature MUST be verified, as specified in section 3.1.5.1.

The server is responsible for providing the expected sequence number for signature validation. The sequence number for the next incoming request is stored in ServerNextReceiveSequenceNumber. The server MUST remember the appropriate sequence number for the response to this request and does so by inserting it into the ServerSendSequenceNumber table with the Mid that identifies the request/response pair. (Mid is as specified in [CIFS] section 3.1.5.)

If the signature on the received packet is incorrect, the server MUST terminate the connection. After verifying that the signature on the current message is correct, the server MUST take the following steps.

```
IF request command EQUALS SMB_COM_NT_CANCEL THEN
    INCREMENT ServerNextReceiveSequenceNumber
ELSE IF request has no response THEN
    INCREMENT ServerNextReceiveSequenceNumber BY 2
ELSE
    SET ServerSendSequenceNumber[Mid] TO
        ServerNextReceiveSequenceNumber + 1
    INCREMENT ServerNextReceiveSequenceNumber BY 2
END IF
```

- SessionExpiry

  If the Uid in the SMB header of the request is 0, the server does not need to check for expiry because a session is not being used for this request. If the Uid in the SMB header of the request is not 0, the server MUST check if the session has expired. If AuthenticationState[Uid] is equal to Expired, the server MUST fail this operation with STATUS_NETWORK_SESSION_EXPIRED unless it is an SMB_COM_SESSION_SETUP_ANDX for session renewal. Upon successful session renewal the server MUST allow all operations as though the session had never expired.

3.3.5.2 Receiving an SMB_COM_NEGOTIATE Request

Processing of an SMB_COM_NEGOTIATE request is handled as specified in [CIFS] with the following additions or clarifications.

- Sending Server Signing Mode

  If the MessageSigningPolicy of the server is Required, the server MUST set both the NEGOTIATE_SECURITY_SIGNATURES_ENABLED and NEGOTIATE_SECURITY_SIGNATURES_REQUIRED bits in the **SecurityMode** field of the SMB_COM_NEGOTIATE response. If the MessageSigningPolicy of the server is Enabled, the server MUST set the NEGOTIATE_SECURITY_SIGNATURES_ENABLED bit in the **SecurityMode** field of the SMB_COM_NEGOTIATE response. If the MessageSigningPolicy of the server is Disabled or DisabledUnlessRequired, the server MUST leave both bits set to FALSE.

- Sending Server Capabilities

  The server MUST set the capabilities that it supports as specified in section 2.2.3 in the **Capabilities** field of the SMB_COM_NEGOTIATE response. This includes setting the new CAP_INFOLEVEL_PASSTHRU and CAP_DYNAMIC_REAUTH capability flags and optionally the CAP_DFS capability flag based on the **GlobalCapabilities** attribute as specified in section 3.3.1.1.

- Generating Extended Security Token

If the client indicated support for extended security by setting SMB_FLAGS2_EXTENDED_SECURITY in the Flags2 field of the SMB header of the SMB_COM_NEGOTIATE request, the server MUST set CAP_EXTENDED_SECURITY in the SMB_COM_NEGOTIATE response. The response MUST take the form specified in section 2.2.3.

The server SHOULD set the **SecurityBlob** of the SMB_COM_NEGOTIATE response to the first GSS token (or fragment thereof) produced by the GSS authentication protocol it is configured to use (GSS tokens are as specified in [RFC2743]). Otherwise, it leaves it empty.

The server MUST initialize its GSS mechanism with the Integrity, Confidentiality, and Delegate options and uses the Server-Initiated variation, as specified in [RFC4178]. The SMB_COM_NEGOTIATE response packet is sent to the client.<168>

- Generating NTLM Encryption Key

  If the client did not set SMB_FLAGS2_EXTENDED_SECURITY in the SMB header of the SMB_COM_NEGOTIATE request, the server MUST generate an SMB_COM_NEGOTIATE response, as specified in [CIFS] section 4.1.1, for the NTLM 0.12 dialect.

  The server creates an NTLM CHALLENGE_MESSAGE, as specified in [NTLM], for server-initiated authentication, extracts the **ServerChallenge** field, and MUST set the **EncryptionKey** field in the SMB_COM_NEGOTIATE response to that value. This value MUST also be stored in the NTLMEncryptionKey.

3.3.5.3 Receiving an SMB_COM_SESSION_SETUP_ANDX Request

Processing of an SMB_COM_SESSION_SETUP_ANDX request is handled as specified in [CIFS] with the following additions or clarifications.

- Storing Client Capabilities

  If ClientCapabilities is equal to 0, the server MUST set ClientCapabilities to the **Capabilities** field received in the SMB_COM_SESSION_SETUP_ANDX request. If ClientCapabilities has already been determined (and is nonzero), the server MUST ignore the capabilities value on subsequent requests.

- Determine Reauth or Continuation of Previous Auth

  If the **Uid** in the SMB header is not 0, the server MUST look up the authentication state for this session, and take the following action based on this state.

  - If AuthenticationState[Uid] is InProgress, this is a continuation of an authentication in progress. This state indicates that the authentication required multiple round trips, and authentication continues.
  - If AuthenticationState[Uid] is Complete or Expired, this is the reauthentication of a user. The server MUST set AuthenticationState[Uid] to InProgress and begin a new authentication for this session. The server MUST prevent any further operations from executing on this session until authentication is complete, failing them with STATUS_NETWORK_SESSION_EXPIRED.
  - If there is no AuthenticationState for the provided Uid, the request MUST be failed with STATUS_INVALID_PARAMETER.

- Extended Security

If CAP_EXTENDED_SECURITY is set in ClientCapabilities, the server MUST handle authentication as defined in this section. Otherwise, it SHOULD continue to the following NTLM authentication section.

The server MUST extract the GSS token, which is the **SecurityBlob** contained in the request, with a length of **SecurityBlobLength**. The server MUST use the configured GSS authentication protocol to obtain the next GSS output token for the authentication protocol exchange. Note that this token can be 0 bytes in length.

If the GSS mechanism indicates an error that is not STATUS_MORE_PROCESSING_REQUIRED, the server MUST fail the client request, returning only an SMB header and propagating the failure code. If a Uid was present in this request, it MUST be removed from the AuthenticationState table. If an entry is present in the SessionKeyEntry for this Uid, it MUST also be removed. The authentication has failed, and no further processing is done on this request. This error response is sent to the client.

If the GSS mechanism indicates success, the server MUST create an SMB_COM_SESSION_SETUP_ANDX response (section 2.2.5). The **SecurityBlob** MUST be set to the output token from the GSS mechanism, and **SecurityBlobLength** is set to the length of the output token. SMB_FLAGS2_EXTENDED_SECURITY is set in the **Flags2** field of the SMB header of the response.

If the request did not specify a Uid in the SMB header of the request, a Uid MUST be generated to represent this user's authentication, as specified in [CIFS] section 2.8, and its value MUST be placed in the **Uid** field of the SMB header of the response.

If the GSS mechanism indicates that the current output token is the last output token of the authentication exchange based on the return code, as specified in [RFC4178], the **Status** field in the SMB header of the response MUST be set to STATUS_SUCCESS, and AuthenticationState[Uid] MUST be set to Completed. Otherwise, the **Status** field in the SMB header of the response MUST be set to STATUS_MORE_PROCESSING_REQUIRED, and AuthenticationState[Uid] MUST be set to InProgress.

Processing for this request continues in the Signing Initialization section below.

- NTLM Authentication

  If CAP_EXTENDED_SECURITY is not set in the ClientCapabilities, the authentication request is processed as defined in this section.

  Upon receipt of the SMB_COM_SESSION_SETUP_ANDX request, the server MUST construct an NTLM AUTHENTICATE_MESSAGE as follows:

  1. The **Signature** field is 8-byte character array that MUST contain the ASCII string ('N', 'T', 'L', 'M', 'S', 'S', 'P', '\0').

  2. The **MessageType** field is set to 3.

  3. The **NtChallengeResponse** field is set to the value of the **CaseSensitivePassword** field from the SMB_COM_SESSION_SETUP_ANDX message.

  4. The **LmChallengeResponse** field is set to the value of the **CaseInsensitivePassword** field from the SMB_COM_SESSION_SETUP_ANDX message.

  5. The **DomainName** field is set from the **PrimaryDomain**, which is determined by the security context of the user initiating the connection to the share.

6. The **UserName** field is set from the **AccountName**, which is determined by the security context of the user initiating the connection to the share.

7. The **WorkstationName** is set to the client name or textual IP address of the client of the connection (as a null-terminated array of 16-bit Unicode characters).

8. EncryptedRandomSessionKey is set to 0 length.

9. Append payload.

The server processes the NTLM AUTHENTICATE_MESSAGE, as specified in [NTLM]. If the authentication fails, an error response MUST be returned to the client propagating the error code, and no further processing is done on this request. If authentication is successful, the server MUST generate a user identifier (Uid) for this user session, as specified in [CIFS] section 2.8, and returns it in the **Uid** field of the SMB header in the successful SMB_COM_SESSION_SETUP_ANDX response.

- Signing Initialization

  If IsSigningActive is FALSE, and the response of the SMB_COM_SESSION_SETUP_ANDX operation contains STATUS_SUCCESS, the server MUST determine if signing can be activated. If bit 0 of the Action field of the SMB_COM_SESSION_SETUP_ANDX response is set, indicating that the user who authenticated did so with the guest account, then signing MUST NOT be activated. Likewise, if the user who authenticated was anonymous (presented no credentials), then signing MUST NOT be activated.

  Otherwise, signing MUST be activated. IsSigningActive MUST be set to TRUE. The server MUST query the authentication protocol (either NTLM or via GSS API) for the session key used in this authentication and store it as ConnectionSigningSessionKey. If CAP_EXTENDED_SECURITY is set in ClientCapabilities, it MUST set ConnectionSigningUseNTLMChallengeResponse to FALSE. If that capability is not set, it MUST set ConnectionSigningUseNTLMChallengeResponse to TRUE and set ConnectionSigningNTLMChallengeResponse to the challenge response received in the SMB_COM_SESSION_SETUP_ANDX request.

  Once these steps are done, the server MUST sign the SMB_COM_SESSION_SETUP_ANDX response. The server follows the steps as specified in section 3.1.4.1, passing in an index number of 1.

  - Protecting Signing Key

    If authentication is successful, the server MUST query the session key from the authentication package (as specified in [NTLM] for implicit NTLM and in [RFC4178] for extended security). The server inserts this value into the SessionKeyEntry table, and MUST set its state to Unavailable.

  - Authentication Expiry

    If the authentication (either NTLM or GSS processing) returned an expiration time, such as a Kerberos ticket time-out, the server MUST set up a Timer to mark the AuthenticationState as Expired when the timeout occurs.

3.3.5.4 Receiving an SMB_COM_TREE_CONNECT_ANDX Request

Processing of an SMB_COM_TREE_CONNECT_ANDX request is handled as specified in [CIFS] with the following additions or clarifications.

- Requesting Extended Information

If the TREE_CONNECT_ANDX_EXTENDED_INFORMATION is set in the **Flags** field of the SMB_COM_TREE_CONNECT_ANDX request, the server MUST respond with the structure as specified in section 2.2.7. The server MUST populate the OptionalSupports bits of the response with the values retrieved from the share. The server MUST calculate the maximal access rights for this response. MaximalShareAccessRights is set to the highest access rights the user has for this share. If the share has no security descriptor in its entry in the ShareList, the value MUST be set to 0xFFFFFFFF. Otherwise, the user initiating the tree connect is evaluated against the security descriptor taken from the share entry, and the maximal access rights for this user are determined and returned in the format as specified in [CIFS] section 2.7. If the share is a DFS share as indicated by the **OptionalSupport** attribute of the share as specified in section 3.3.1.2, it MUST set the SMB_SHARE_IN_DFS bit of the **OptionalSupport** field in the SMB_COM_TREE_CONNECT_ANDX response.

In a similar fashion, GuestMaximalAccessRights MUST be set to the highest access rights the guest account has on this share. If the share has no access limits enforced, this value also MUST be 0xFFFFFFFF. Otherwise, the guest user is evaluated against the security descriptor taken from the entry in the ShareList, and the maximal access rights for guests are determined and returned in the format as specified in [CIFS] section 2.7. If the system does not support the guest account, it MUST set GuestMaximalAccessRights to 0.

- Requesting Signing Key Protection

  If the client has set the TREE_CONNECT_ANDX_EXTENDED_SIGNATURE bit in the **Flags** field of the SMB_COM_TREE_CONNECT_ANDX request, the server MUST hash the session key of the calling user. This protects the key used for signing by making it unavailable to server-side applications.

  The one-way hash MUST be done on the session key as specified in section 3.2.5.4. The hash done by the server is identical to that performed by the client described there.

  After the session key has been hashed, or if the TREE_CONNECT_ANDX_EXTENDED_SIGNATURE bit is not set, the server MUST set the state in the SessionKeyEntry table for this session to Available, allowing applications to query the session key.

3.3.5.5 Receiving an SMB_COM_NT_CREATE_ANDX Request

Other than the changes listed below, the request is processed as specified in [CIFS] section 4.2.1.

On a successful create, if the NT_CREATE_REQUEST_EXTENDED_RESPONSE flag was set in the **Flags** field of the SMB_COM_NT_CREATE_ANDX request, the server MAY choose to send an extended response, as defined in section 2.2.9, if it wants the client to leverage the new capabilities. Otherwise, it sends the response as specified in [CIFS] section 4.2.1. If it chooses to send the new request, it MUST set the **WordCount** to 42. The server MUST query the file attributes and use them to set the FileStatusFlags in the response. The server MAY choose to fill in the VolumeGUID and **FileId** fields if it wants the client to leverage the new capabilities. Otherwise, they MUST be set to 0. Finally, the server MUST calculate the maximal access the calling user has on the file and place it in the **MaximalAccessRights** field of the response. If the file has no security applied, the field MUST be set to 0xFFFFFFFF. Likewise, it MUST query the maximal access a Guest user has on this file, and place it in the **GuestMaximalAccessRights** field of the response. The Server MUST use the authenticated session as specified in section 3.3.5.3 and the impersonation level to impersonate the client. If the server is unable to perform impersonation as specified by **ImpersonationLevel**, the server MUST fail the request with STATUS_BAD_IMPERSONATION_LEVEL.<169>

3.3.5.5.1 Scanning the Path for a Previous Version Token

If the request has SMB_FLAGS2_REPARSE_PATH set in the **Flag2** field of the SMB header, the server MUST perform a parse of the path checking for previous version tokens. If the flag is not set, the server MAY choose to parse the path anyway. The path name is searched for a directory or final element matching the @GMT token format. If this element is found, it is removed from the path and the time stamp is evaluated. The request then implies an attempt to open this path at the time specified by the time stamp. For example, opening a version for file \\server\mydocs\reviews\feb01.doc at 2:44:00 PM on 3/30/01 UTC is specified in the following format.

```
\\server\mydocs\reviews\@GMT-2001.03.30-14.44.00\feb01.doc
```

The same technique can be used to build a path that represents a previous version of a directory as opposed to a file. In this case, the token can go either as an intermediate path or as the final path element.

Version for directory \\server\mydocs\reviews at 2:44:00 PM on 3/30/01 UTC is specified in the following format.

```
\\server\mydocs\reviews\@GMT-2001.03.30-14.44.00
```

The server MUST do the reverse of these transformations to split the path into its target file/directory and its time stamp. If a version of the file does not exist for the given time stamp, the server MUST fail the operation with STATUS_OBJECT_NAME_NOT_FOUND. If the file does exist, processing continues as normal, except the execution is against the view of the volume at the time of the time stamp provided.

This same logic MUST be applied to all operations as specified in [CIFS] that take a file path as a parameter.<170>

3.3.5.6 Receiving an SMB_COM_OPEN_ANDX Request (Obsolete)

Other than the changes listed below, the request is processed as specified in [CIFS] section 5.8.

The file path MUST be parsed for previous version tokens, as specified in section 3.3.5.5.

On a successful open, if the SMB_OPEN_EXTENDED_RESPONSE flag was set in the **Flags** field of the SMB_COM_OPEN_ANDX request, the server MAY choose to send an extended response, as defined in section 2.2.11. Otherwise, it sends the response as specified in [CIFS] section 5.8. If the server chooses to send the new response, it MUST set the **WordCount** to 19. The server MUST calculate the maximal access the calling user has on the file and place it in the **MaximalAccessRights** field of the response. If the file has no security applied, the field MUST be set to 0xFFFFFFFF. Likewise, it MUST query the maximal access a Guest user has on this file, and place it in the **GuestMaximalAccessRights** field of the response.

The response is sent to the client.<171>

3.3.5.7 Receiving an SMB_COM_READ_ANDX Request

Other than the changes listed below, the request is processed as specified in [CIFS] section 4.2.4.

If CAP_LARGE_READX is set in ClientCapabilities, it is possible that the **MaxCount** sent is larger than the negotiated buffer size. In this case, the server MUST perform the read for as many bytes as it can support up to the received **MaxCount** in the received SMB_COM_READ_ANDX request. If

WordCount of the request is 0x0c (12), the values in **MaxCountHigh** MUST be treated as the two most significant bytes of the read length, combined with **MaxCount** to create a 32-bit read length (as specified in section 3.2.4.4) with one exception. If **MaxCountHigh** is set to 0xFF, the value MUST be ignored, and only the length received in **MaxCount** MUST be used.

It is acceptable to return fewer bytes than requested by the client, but the server SHOULD allow for reads of at least 64 KB. If the number of bytes read is greater to or equal to 64 KB, the server MUST set the less significant 16-bits of the length in the DataCount field in the response, and the more significant 16-bits of the length in the **DataCountHigh** field, which is the first 2 bytes of Reserved[5], as specified in [CIFS] section 4.2.4.<172>

3.3.5.8 Receiving an SMB_COM_WRITE_ANDX Request

Other than the changes listed below, the request is processed as specified in [CIFS] section 4.2.5.

If CAP_LARGE_WRITEX is set in ClientCapabilities, it is possible that the **DataLength** sent is larger than the negotiated buffer size. In this case, the server MUST perform the write for as many bytes as indicated by the **DataLength** field in the received SMB_COM_WRITE_ANDX request.

3.3.5.9 Receiving a TRANS2_FIND_FIRST2 Request

Other than the changes listed below, the request is processed as specified in [CIFS] section 4.3.3.

The file path MUST be parsed for previous version tokens, as specified in section 3.3.5.5.

The server MAY allow for the new information levels, as specified in section 2.2.13.13. If the server does not support the new information levels, it MUST fail the operation with STATUS_NOT_SUPPORTED.<173>

If the FileName of the TRANS_FIND_FIRST2 request contains the search pattern @GMT-*, and the requested InformationLevel is SMB_FIND_FILE_BOTH_DIRECTORY_INFO, the server MAY choose to return an enumeration of previous versions valid for the share. It does so by manufacturing a file entry for each previous version, as defined in section 2.2.13.14.3. If the server chooses not to, the enumeration MUST be processed as a normal FindFirst operation.<174>

3.3.5.10 Receiving a TRANS2_FIND_NEXT2 Request

Other than the changes listed below, the request is processed as specified in [CIFS] section 4.3.4.

If the query is started using one of the new InformationLevels as specified in section 2.2.13.13, the same information level structure MUST be used for the return of subsequent entries in the enumeration continuation.

Likewise, a query for previous version information that is started, as specified in section 3.3.5.9, MUST be continued at the client's request with further entries generated, as defined there.

3.3.5.11 Receiving a TRANS2_QUERY_FILE_INFORMATION Request

Other than the changes listed below, the request is processed as specified in [CIFS] section 4.2.15.

If the InformationLevel in the TRANS2_QUERY_FILE_INFORMATION request is greater than or equal to 0x3e8 (1,000), and the server returned the CAP_INFOLEVEL_PASSTHRU capability, the server MUST decrement the InformationLevel by 0x3e8 (1,000), treating the value as little-endian, and evaluate it as a native Windows NT information level, as specified in [MS-FSCC] section 2.4. This information MUST be queried from the file system and returned in the Data section of the response,

as specified in [CIFS] section 4.2.15. If the server file system does not support this InformationLevel, it MUST fail the request with STATUS_NOT_SUPPORTED.

If the server did not return CAP_INFOLEVEL_PASSTHRU in the capabilities in Negotiate, and the InformationLevel is 0x3e8 or greater, it MUST fail this request with STATUS_INVALID_PARAMETER.

3.3.5.12 Receiving a TRANS2_QUERY_PATH_INFORMATION Request

Other than the changes listed below, the request is processed as specified in [CIFS] section 4.2.14.

Because this is a path based operation, the file path MUST be parsed for previous version tokens, as specified in 3.3.5.5.

If the InformationLevel in the TRANS2_QUERY_PATH_INFORMATION request is greater than or equal to 0x3e8 (1,000), and the server returned the CAP_INFOLEVEL_PASSTHRU capability, the server MUST decrement the InformationLevel by 0x3e8 (1,000), treating the value as little-endian, and evaluate it as a native Windows NT information level, as specified in [MS-FSCC] section 2.4. This information MUST be queried from the file system and returned in the Data section of the response, as specified in [CIFS] section 4.2.14. If the server file system does not support this InformationLevel, it MUST fail the request with STATUS_NOT_SUPPORTED.

If the server did not return CAP_INFOLEVEL_PASSTHRU in the capabilities in Negotiate, and the InformationLevel is 0x3e8 or greater, it MUST fail this request with STATUS_INVALID_PARAMETER.

3.3.5.13 Receiving a TRANS2_SET_FILE_INFORMATION Request

Other than the changes listed below, the request is processed as specified in [CIFS] section 4.2.17.

If the InformationLevel in the TRANS2_SET_FILE_INFORMATION request is greater than or equal to 0x3e8 (1,000), and the server returned the CAP_INFOLEVEL_PASSTHRU capability, the server MUST decrement the InformationLevel by 0x3e8 (1,000), treating the value as little-endian, and evaluate it as a native Windows NT information level, as specified in [MS-FSCC] section 2.4. This is the information received in the requests Data buffer, and it is being applied to the file. If the server file system does not support this InformationLevel, it MUST fail the request with STATUS_NOT_SUPPORTED. Otherwise, it MUST attempt to apply the attributes to the target file, and return the success or failure code in the response.

If the server did not return CAP_INFOLEVEL_PASSTHRU in the capabilities in Negotiate, and the InformationLevel is 0x3e8 or greater, it MUST fail this request with STATUS_INVALID_PARAMETER.

3.3.5.14 Receiving a TRANS2_SET_PATH_INFORMATION Request

Other than the changes listed below, the request is processed as specified in [CIFS] section 4.2.16.

Because this is a path-based operation, the file path MUST be parsed for previous version tokens, as specified in section 3.3.5.5.

If the InformationLevel in the TRANS2_SET_PATH_INFORMATION request is greater than or equal to 0x3e8 (1,000), and the server returned the CAP_INFOLEVEL_PASSTHRU capability, the server MUST decrement the InformationLevel by 0x3e8 (1,000), treating the value as little-endian, and evaluate it as a native Windows NT information level, as specified in [MS-FSCC] section 2.4. This is the information received in the requests Data buffer, and it is being applied to the file. If the server file system does not support this InformationLevel, it MUST fail the request with STATUS_NOT_SUPPORTED. Otherwise, it MUST attempt to apply the attributes to the target file and return the success or failure code in the response.

If the server did not return CAP_INFOLEVEL_PASSTHRU in the capabilities in Negotiate, and the InformationLevel is 0x3e8 or greater, it MUST fail this request with STATUS_INVALID_PARAMETER.

3.3.5.15 Receiving a TRANS2_QUERY_FS_INFORMATION Request

Other than the changes listed below, the request is processed as specified in [CIFS] section 4.1.6.

If the InformationLevel in the TRANS2_QUERY_FS_INFORMATION request is greater than or equal to 0x3e8 (1,000), and the server returned the CAP_INFOLEVEL_PASSTHRU capability, the server MUST decrement the InformationLevel by 0x3e8 (1,000), treating the value as little-endian, and evaluate it as a native Windows NT information level, as specified in [MS-FSCC] section 2.5. This information MUST be queried from the file system and returned in the Data section of the response, as specified in [CIFS] section 4.2.15. If the server file system does not support this InformationLevel, it MUST fail the request with STATUS_NOT_SUPPORTED.

If the server did not return CAP_INFOLEVEL_PASSTHRU in the capabilities in Negotiate, and the InformationLevel is 0x3e8 or greater, it MUST fail this request with STATUS_INVALID_PARAMETER.

3.3.5.16 Receiving a TRANS2_SET_FS_INFORMATION Request

Processing for this operation is identical to that of TRANS2_SET_FILE_INFORMATION, as specified in [CIFS] section 4.2.15, except the attributes are being applied to the file system that the share resides on instead of a file.

If the InformationLevel in the TRANS2_SET_FS_INFORMATION request is greater than or equal to 0x3e8 (1,000), and the server returned the CAP_INFOLEVEL_PASSTHRU capability, the server MUST decrement the InformationLevel by 0x3e8 (1,000), treating the value as little-endian, and evaluate it as a native Windows NT information level, as specified in [MS-FSCC] section 2.5. This is the information received in the requests Data buffer, and it is being applied to the file. If the server file system does not support this InformationLevel, it MUST fail the request with STATUS_NOT_SUPPORTED. Otherwise, it MUST attempt to apply the attributes to the target file and return the success or failure code in the response.

If the server did not return CAP_INFOLEVEL_PASSTHRU in the capabilities in Negotiate, it MUST fail this request with STATUS_INVALID_PARAMETER.

3.3.5.17 Receiving an NT_TRANS_IOCTL Request

Other than the changes or clarifications listed below, the request is processed as specified in [CIFS] section 4.6.1.

The FunctionCode is taken from the NT_TRANS_IOCTL request, and the input buffer is contained in the Data buffer of the request. The server MUST pass the I/O control request to the underlying file system unless it is one of the server-processed FSCTLs listed below. If an error is returned, the server MUST return this error to the client in the response. If the operation is successful, the server MUST return the output buffer in the Data buffer of the NT_TRANS_IOCTL response. A list of common Windows NT native FSCTLs is as specified in [MS-FSCC] section 2.3.

3.3.5.17.1 Receiving an FSCTL_ENUMERATE_SNAPSHOTS Request

This is a request to enumerate available previous versions for the share in question. If the server does not support previous versions, it MUST fail this request with STATUS_NOT_SUPPORTED.

Otherwise, the server MUST return an enumeration of available previous versions, as specified in section 2.2.14.8.1. The **NumberOfSnapshots** MUST contain the total number of previous versions available for the volume, and **NumberOfSnapshotsReturned** contains the number of entries

returned in this enumeration. The value returned in **SnapShotArraySize** MUST be the size required to receive all available previous versions. If the **MaxDataCount** of the request does not allow for the return of **SnapShotArraySize**, the server MUST fail the request with STATUS_INVALID_PARAMETER.

The response is sent to the client.

3.3.5.17.2 Receiving an FSCTL_QUERY_RESUME_KEY Request

This is a request for an opaque file identifier for use in an FSCTL_SRV_COPYCHUNK operation. The server MUST provide a 24-byte value that is used to uniquely identify the open of the file that this operation is executed against. The server MUST set **ContextLength** to 0. If the server does not support this operation, it MUST fail the request with STATUS_NOT_SUPPORTED. Otherwise, the resulting 24-byte value is returned in the response, as specified in section 2.2.14.8.2.

3.3.5.17.3 Receiving an FSCTL_SRV_COPYCHUNK Request

This is a request for a server-side data copy, as specified in section 2.2.14.7.3. If the server does not support this operation, it MUST fail the request with STATUS_NOT_SUPPORTED.

Otherwise, the server MUST identify the source file based on the **Key** field of the FSCTL_SRV_COPYCHUNK request. This **Key** is a value that was returned by the server from an FSCTL_QUERY_RESUME_KEY operation. If the **Key** is not valid, or does not represent a file opened for read-data access, the server MUST fail the operation with STATUS_ACCESS_DENIED. Likewise, the target file MUST be specified by the Fid in the SMB_COM_NT_TRANSACTION request. If the target file is not opened for write-data access, the server MUST fail the operation with STATUS_ACCESS_DENIED.

The server MUST validate that the amount of data to be written is within the server's configured bounds. If either the number of chunks is too high, or the maximum write size of any individual chunk is too large, or the total write size of all chunks in the request is too large, the server MUST fail the operation with STATUS_INVALID_PARAMETER and return a response, as specified in section 2.2.14.8.3. (**ChunksWritten** MUST contain the maximum number of chunks that can be written in a single operation. **ChunkBytesWritten** MUST contain the maximum number of bytes that can be written in a single chunk. And **TotalBytesWritten** MUST contain the total amount of data that can be written in an FSCTL_SRV_COPYCHUNK operation.)

If both the source and the target file are appropriate, the server MUST iterate through the data ranges specified in the request, reading data from the source offset of the source file and writing it to the target offset of the target file. The server MUST stop if an error is encountered and set the response parameters, as specified in section 2.2.14.8.3, to indicate how much data was successfully written.

The server MAY restrict the amount of processing that can be done for a single FSCTL_SRV_COPYCHUNK operation by the amount of time it has to process. If this exceeds the server side configured timeout period, it MUST fail the request with STATUS_IO_TIMEOUT and return the relevant number of chunks written, as specified in section 2.2.14.8.3.<175>

3.3.5.18 Receiving a GET_DFS_REFERRALS Request

If the DFS server is not active, the request MUST be failed with a STATUS_NO_SUCH_DEVICE error. Otherwise, the SMB server MUST pass to the DFS server the following:

- The IP address of the client.

- The maximum size of the response data buffer that will be accepted by the client as described by the **MaxDataCount** field in the TRANSACT2 request.
- A response data buffer to be filled in by the DFS server.
- The parameter buffer of the transact2 request and its length in bytes.

The response data buffer returned by the DFS server after it processes the request MUST be copied into the data block of the TRANSACT2 response and returned to the client. The TotalDataCount field of the TRANSACT2 response MUST be set to the size in bytes of the response data buffer. How to format a TRANSACT2 request and response is as specified in [CIFS] section 3.13.1.

3.3.5.19 Receiving a NT_TRANS_QUERY_QUOTA_INFO Request

Upon receiving an SMB_COM_NT_TRANSACTION request with the subcommand of NT_TRANS_QUERY_QUOTA_INFO, the server MUST enumerate quota information on the volume on which the share resides. If the volume does not support quotas, the server MUST return STATUS_NOT_SUPPORTED.

The server MUST return as much of the available quota information as can fit in the maximum response buffer size denoted by **MaxDataCount**. The format of the request determines what entries need to be returned, as specified in section 2.2.14.3. The server MUST place the quota information in the response, as specified in section 2.2.14.4, and send the response back to the client.

3.3.5.20 Receiving a NT_TRANS_SET_QUOTA_INFO Request

Upon receiving an SMB_COM_NT_TRANSACTION request with the subcommand of NT_TRANS_SET_QUOTA_INFO, the server MUST apply the provided quota information to the volume on which the share resides. If the volume does not support quotas, the server MUST return STATUS_NOT_SUPPORTED.

The server MUST apply the quota information provided in the Data buffer of the request. The resulting success or error received from the file system is returned in the response.

3.3.5.21 Receiving a TRANS_SET_NMPIPE_STATE Request

For processing the SMB_COM_TRANSACTION request with a TRANS_SET_NMPIPE_STATE subcommand, the server MUST apply the state provided in the Parameter buffer of the request to the named pipe specified by the Fid contained in the second Setup WORD, as specified in section 2.2.12.1. The success or failure code of this operation is returned in the response.

3.3.5.22 Receiving a TRANS_QUERY_NMPIPE_INFO Request

For processing the SMB_COM_TRANSACTION request with a TRANS_QUERY_NMPIPE_STATE subcommand, the server MUST query the named pipe specified by the Fid contained in the second Setup WORD for the relevant information, as specified in section 2.2.12.8. This information is returned in the Data portion of the SMB_COM_TRANSACTION response, as specified in that section.

3.3.5.23 Receiving a TRANS_PEEK_NMPIPE Request

For processing the SMB_COM_TRANSACTION request with a TRANS_PEEK_NMPIPE subcommand, the server MUST read data from the pipe specified by the Fid contained in the second Setup WORD without removing the data from the pipe queue. The amount of data to be read is specified in the **MaxDataCount** value of the incoming request. The server returns the data bytes read in the Data buffer in the response, as specified in section 2.2.12.10. The number of bytes available to be read is

indicated in the response (ReadDataAvailable) and can be less than the **MaxDataCount** if that many bytes are not in the pipe.

3.3.5.24 Receiving a TRANS_TRANSACT_NMPIPE Request

For processing the SMB_COM_TRANSACTION request with a TRANS_TRANSACT_NMPIPE subcommand, the server MUST write data to the pipe specified by the Fid contained in the second Setup WORD. The amount of data to be written is specified in the **TotalDataCount** value of the incoming request. The server MUST read data from the pipe with the number of bytes to be read specified in the **MaxDataCount** value of the incoming request. The server MUST return the data bytes read in the Data buffer in the response, as specified in section 2.2.12.12. If the pipe is a message mode pipe, and the entire message is not read, the server MUST return a status of STATUS_BUFFER_OVERFLOW. Otherwise, the server MUST return STATUS_SUCCESS.

3.3.5.25 Receiving a TRANS_QUERY_NMPIPE_STATE

For processing the SMB_COM_TRANSACTION request with a TRANS_QUERY_NMPIPE_STATE subcommand, the server MUST query the named pipe specified by the Fid contained in the second Setup WORD for the relevant information, as specified in section 2.2.12.6. This information is returned in the Data portion of the SMB_COM_TRANSACTION response, as specified in that section.

3.3.5.26 Processing a RAP Transaction Request

When an SMB server receives a SMB_COM_TRANSACTION request with a **SetupCount** of 0 on the IPC$ share, it MUST process the request as follows:

If a RAP server is not active as specified in section 3.3.4.6, it MUST fail the request with a STATUS_NOT_IMPLEMENTED error. Otherwise, the following data from the transact request MUST be provided to the RAP server.

1. The Name/IP address of the client.

2. The security context for the user as indicated by the UID in the SMB header.

3. The transaction name from the SMB_COM_TRANSACTION request.

4. The data contained in the parameter block of the SMB_COM_TRANSACTION request and the length in bytes of the parameter buffer (**TotalParameterCount** in the transact request). The SMB server must also pass the maximum size of the parameter block in the transact response that will be accepted by the client as described by the **MaxParameterCount** field in the transact request and a buffer for the response parameters. The response parameter buffer filled in by the RAP server MUST be returned to the client via the parameter block of SMB_COM_TRANSACTION response. The **TotalParameterCount** of the transaction response MUST be set to the number of bytes in the response parameter buffer.

5. The data contained in the data block of the SMB_COM_TRANSACTION request and the length in bytes of the data buffer (**TotalDataCount** in the transact request). The SMB server must also pass the maximum size of the data block in the transact response that will be accepted by the client as described by the **MaxParameterCount** field in the transact request and a buffer for the response data. The response data buffer filled in by the RAP server MUST be returned to the client via the data block of SMB_COM_TRANSACTION response. The **TotalDataCount** of the transaction response MUST be set to the number of bytes in the response data buffer.

The format of SMB_COM_TRANSACTION requests and responses is specified in [CIFS] section 3.13.1.

3.3.6 Timer Events

When the Authentication Expiration Timer expires, the server scans the active authentication tokens and marks the Authentication state for those whose expiry time has passed. It MUST set AuthenticationState[Uid] = Expired.

3.3.7 Other Local Events

No other local events are required beyond what is required by the CIFS Protocol, as specified in [CIFS].

4 Protocol Examples

The following sections describe common scenarios to illustrate the function of the SMB Protocol.

4.1 Extended Security Authentication

The following diagram depicts the protocol message sequence for a multi-phase extended security exchange and previous versions enumeration and access on the share root folder.

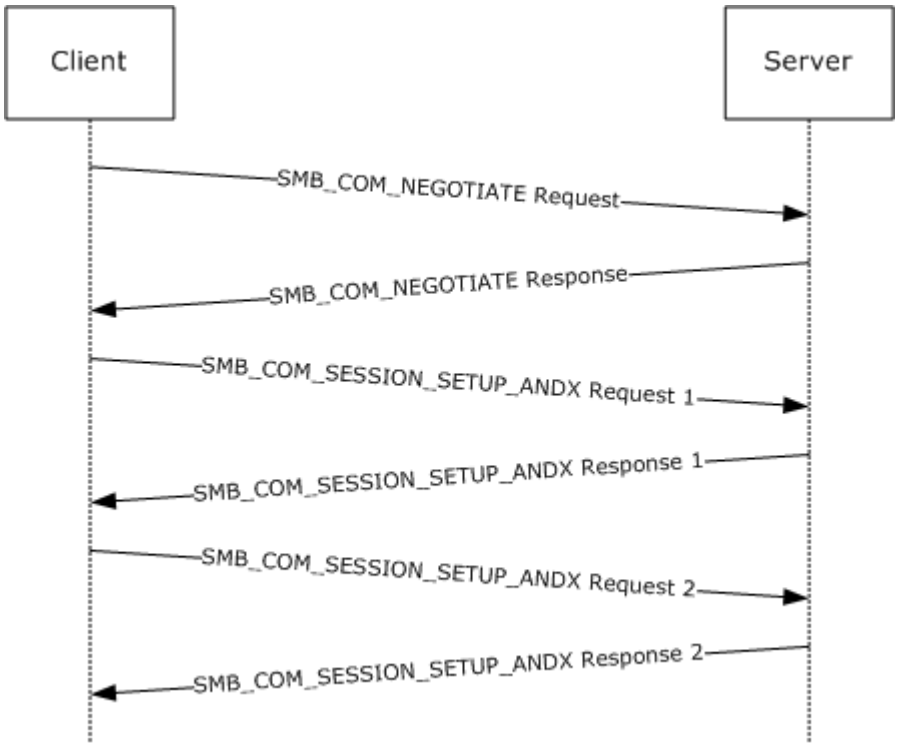


Figure 4: User authentication and session establishment sequence

Descriptions of the fields in this example are specified in [CIFS] and section 2.2.3. Fields shown and highlighted in bold text are relevant to this extension. It is assumed the client has successfully established a network connection with the server.

The client initiates the first message with a SMB_COM_NEGOTIATE request, as specified in [CIFS]. The client specifies extended security negotiation in the header **Flags2** field. It also includes NT LM 0.12 in the dialect strings list. The server constructs an extended SMB_COM_NEGOTIATE response packet that is denoted by the *WordCount* parameter. The server returns dialect index, its capabilities, GUID value, and the initial security blob obtained, as specified in [RFC4178] and defined above.

FRAME 1. Client negotiate request

```
Client -> Server: Command = SMB_COM_NEGOTIATE
Flags2 Summary = 51207 (0xC807)
  1100 1000 0000 0111
  .... 1... .... .... = Extended security negotiation is supported
Dialect Strings
```

```
PC NETWORK PROGRAM 1.0
LANMAN1.0
Windows for Workgroups 3.1a
LM1.2X002
LANMAN2.1
NT LM 0.12
```

FRAME 2. Server negotiate response

```
Server -> Client: Command = SMB COM NEGOTIATE
NT status code = 0x0, STATUS SUCCESS
Word count = 17
Protocol Index = 5  (NT LM 0.12)
Capabilities = 2147607549 (0x8001F3FD)
 1000 0000 0000 0001 1111 0011 1111 1101
 ....  ....  ....  ....  ..1.  ....  ....  ....  = Supports Pass-Thru levels
 1...  ....  ....  ....  ....  ....  ....  ....  = Supports extended security
Server GUID = 01 B3 1E 23 07 2A A4 4D A1 9F B6 69 F0 45 71 90
Security Blob in payload
```

The client uses the initial security blob returned by the server along any user credential information to obtain its security blob, as specified in [RFC4178] and defined in section 3.2.4.2.3. The resulting security blob is sent to the server as part of the SMB_COM_SESSION_SETUP_ANDX extended request. The client also sends its capabilities and zero Uid to mark the start of a new session setup exchange. The server verifies that the client is requesting extended security by checking **Flags2** and **Capabilities** fields in the request, accepts as input the client security blob, and processes it, as specified in [RFC4178]. In this case, the security package requires more processing and returns a second security blob to be returned to the client. Also, the server allocates a new Uid and associates it with this session setup exchange.

Note Extended security MAY require multiple request and response exchanges between client and server to complete. The Uid is defined by the server on first response to an extended session setup and used for the lifetime of the session.

FRAME 3. Client request for extended session setup

```
Client -> Server: Command = SMB COM SESSION SETUP ANDX
Header: Tid = 0x0000 Mid = 0x0070 Uid = 0x0000
Flags2 Summary = 51207 (0xC807)
 1100 1000 0000 0111
 ....  1...  ....  ....  = Supports extended security
Word count = 12
Capabilities = 0xA0000000
1010 0000 0000 0000 0000 0000 0000 0000
..1.  ....  ....  ....  ....  ....  ....  ....  = Supports dynamic reauth
1...  ....  ....  ....  ....  ....  ....  ....  = Requests extended security
Security Blob Length = 74 (0x4A)
Security Blob in payload
```

FRAME 4. Server response with session setup continuation

```
Server -> Client: Command = SMB_COM_SESSION_SETUP_ANDX
NT status code = 0xC0000016, STATUS_MORE_PROCESSING_REQUIRED
Header: Tid = 0x0000 Mid = 0x0070 Uid = 0x0802
Flags2 Summary = 51207 (0xC807)
  1100 1000 0000 0111
  .... 1... .... .... = Extended security negotiation is supported
Security Blob Length = 349 (0x15D)
            Security Blob in payload
```

The client accepts as input the server security blob and processes it, as specified in [RFC4178], and its output is returned to the server along with the Uid. The server uses the **Uid** value to associate this request with the pending session establishment. The server processes this request, as specified in [RFC4178], and receives a success result. At this point, the SMB_SESSION_SETUP_ANDX exchange is complete because the status code is not equal to STATUS_MORE_PROCESSING. The final security blob is returned with the success indication.

FRAME 5. Client session setup request continuation

```
Client -> Server: Command = SMB_COM_SESSION_SETUP_ANDX
Header: Tid = 0x0000 Mid = 0x0080 Uid = 0x0802
Flags2 Summary = 51207 (0xC807)
 1100 1000 0000 0111
  .... 1... .... .... = Extended security negotiation is supported
Word count = 12
Security Blob Length = 226 (0xE2)
            Security Blob in payload
```

FRAME 6. Server response with session setup completion

```
Server -> Client: Command = SMB_COM_SESSION_SETUP_ANDX
NT status code = 0x0, STATUS_SUCCESS
Header: Tid = 0x0000 Mid = 0x0080 Uid = 0x0802
Security Blob Length = 9 (0x9)
Security Blob in payload
```

At this point, the client has been successfully authenticated.

4.2 Previous File Version Enumeration

The following example shows how the client accesses a previous version of the share root folder. It is assumed that the client has already authenticated, established a tree connection to the target share, and opened a handle to the root directory, as specified in [CIFS]. Thus, frame 1 is not truly the first frame for the connection, but is referred to as the starting point for this operation.

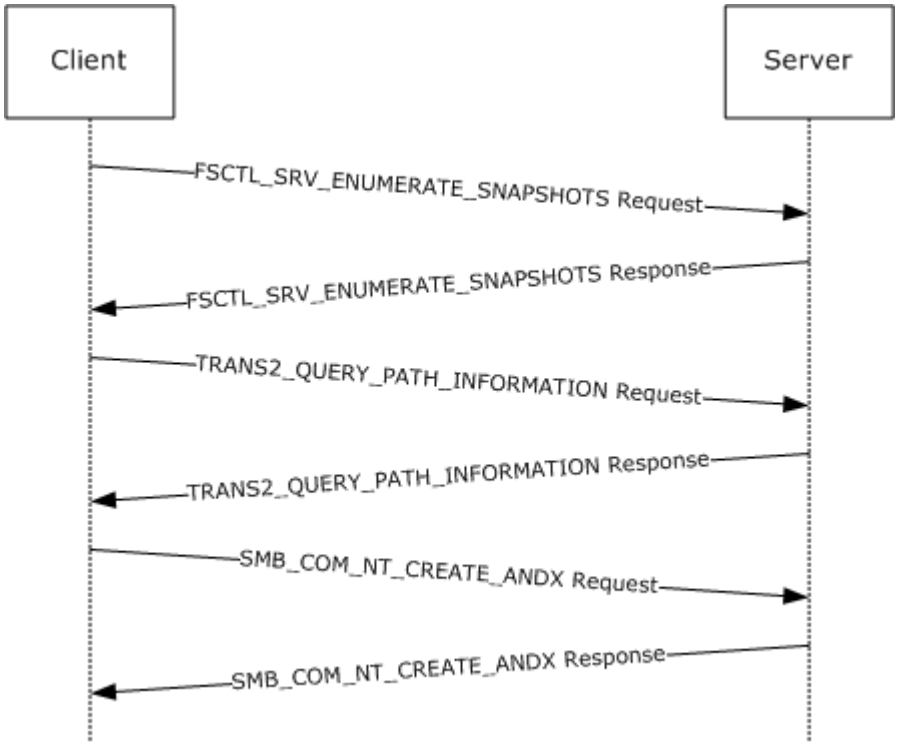


Figure 5: Previous file version enumeration sequence

The first step is to enumerate the list of available snapshots on the server with the FSCTL_SRV_ENUMERATE_SNAPSHOT command. The client requests the list of snapshots available on the server by using the root handle Fid. The server returns the list of snapshots in the format defined above. In this example, the server has one snapshot total for the root folder, the payload contains one snapshot string, the payload size is 0x34 bytes, and the snapshot name is @GMT-2006.04.26-04-08-27. The last 2 bytes of the payload are the snapshot strings 16-bit Unicode NULL delimiter.

FRAME 1. Client request FSCTL_SRV_ENUMERATE_SNAPSHOTS

```
Client -> Server: Command = SMB_COM_NT_TRANSACT
NT IOCTL Function Code 0x00144064 FSCTL_SRV_ENUMERATE_SNAPSHOTS
File ID (Fid) = 16391 (0x4007)
```

FRAME 2. Server response with list of snapshots

```
Server -> Client: Command = SMB COM NT TRANSACT
NT status code = 0x0, STATUS SUCCESS
Payload contained in Data buffer as defined in section 3.1.5.4:
00090:        01 00 00 00 01 00 00 00 34 00 00 00 40 00   ..........4...@.
000A0:  47 00 4D 00 54 00 2D 00 32 00 30 00 30 00 36 00   G.M.T.-.2.0.0.6.
000B0:  2E 00 30 00 34 00 2E 00 32 00 36 00 2D 00 30 00   ..0.4...2.6.-.0.
000C0:  34 00 2E 00 30 00 38 00 2E 00 32 00 37 00 00 00   4...0.8...2.7...
000D0:  00 00
```

The client uses standard SMB commands to access the snapshot. The client also indicates in the header Flags2 that the name in the request is tokenized with previous version information. This indicates to the server that the client is accessing a previous version of the path. The server processes the request and returns the path information for the snapshot directory as opposed to the current directory.

FRAME 3. Client requests path information for snapshot 2006/04/26 04:08:27 AM

```
Client -> Server: Command = SMB_COM_NT_TRANSACT
Flags2 Summary = 52231 (0xCC07)
  1100 1100 0000 0111
  ....  .1..  ....  ....  = File name is tokenized with Previous
Version Information
Transact2 function = Query path info
File name =\@GMT-2006.04.26-04.08.27
00080:                                      5C 00 40 00
……............\.@.
00090:  47 00 4D 00 54 00 2D 00 32 00 30 00 30 00 36 00   G.M.T.-.2.0.0.6.
000A0:  2E 00 30 00 34 00 2E 00 32 00 36 00 2D 00 30 00   ..0.4...2.6.-.0.
000B0:  34 00 2E 00 30 00 38 00 2E 00 32 00 37 00 00 00   4...0.8...2.7...
```

FRAME 4. Server response with snapshot path information

```
Server -> Client: Command = SMB_COM_NT_TRANSACT
NT status code = 0x0, STATUS_SUCCESS
Data bytes = 40 (0x28)
Payload contains path information for specified snapshot version
```

Similar to the query path exchange, the client specifies the previous version of the root folder in an open request. The server processes the request and returns a Fid for the specified previous version of the path.

FRAME 5. Client open request for version 2006/04/26 04:08:27 AM on "\"

```
Client -> Server: Command = SMB_COM_NT_CREATE_ANDX
Flags2 Summary = 52231 (0xCC07)
  1100 1100 0000 0111
  ....  .1..  ....  ....  = File name is tokenized with Previous
Version Information
Create Disposition = Open:  If exist, Open, else fail
            File name =\@GMT-2006.04.26-04.08.27
```

FRAME 6. Server open root folder and returns Fid

```
Server -> Client: Command = SMB_COM_NT_CREATE_ANDX
NT status code = 0x0, STATUS_SUCCESS
File ID (Fid) = 16392 (0x4008)
```

```
Create Action = File Opened
```

These similar steps can be used to open a file as opposed to a directory on a remote volume. In that case, the @GMT token is contained in the relative path, such as \directory\@GMT-2006.04.26-04.08.27\file.txt. This path can be used to query attributes or to open the file with the resulting Fid used to read its contents.

Likewise, the @GMT path in the example can be used as part of a TRANS2_FIND_FIRST2 and TRANS2_FIND_NEXT2 to enumerate the contents of the volume at the time of the snapshot.

4.3 Message Signing Example

The following is the sequence of events related to SMB message authentication. In the following scenario, as specified in [RFC4178], authentication is used between the client and the server. The client and server are both configured not to require SMB signing; however, both are capable of using SMB signing. This also applies to Figure 4 in section 4.1; however, the parameters significant to signing negotiation are called out.

1. The client sends an SMB_COM_NEGOTIATE request to the server.

```
Client -> Server: SMB: C negotiate, Dialect = NTLM 0.12
SMB Flags2 contains 0xC853
  1...  ....  ....  ....  = Unicode Strings: Strings are Unicode
  .1..  ....  ....  ....  = Error Code Type: Error codes are NT error codes
  ..0.  ....  ....  ....  = Execute-Only Reads: Do not permit reads if execute-only
  ...0 ....  ....  ....  = Dfs: Do not resolve pathnames with Dfs
  ....  1...  ....  ....  = Extended security negotiation is supported
  ....  ....  .1..  ....  = Long Names Used
  ....  ....  ....  .0..  = Security signatures are not supported
  ....  ....  ....  ..1.  = Extended Attributes: Extended attributes are supported
  ....  ....  ....  ...1 = Long Names Allowed
Security Signature is not set (the value is 00 00 00 00 00 00 00
00).
SECURITY_SIGNATURE: Bit2  (not set)
```

No **SecuritySignature** is generated at this stage.

2. The client receives an SMB_COM_NEGOTIATE response SMB from the server.

```
Server -> Client: SMB: R negotiate, Dialect # = 5
SMB Flags2 contains 0xC853
Binary: 00000000 00000000 11001000 01010011
                              ^      ^  ^

SECURITY_SIGNATURE: Bit2: (not set)
Security Signature is not set (the value is  00 00 00 00 00 00 00 00).
```

No **SecuritySignature** is generated at this stage.

3. The client builds an SMB_COM_SESSION_SETUP_ANDX request SMB and sends it the server.

In the SessionSetupAndX SMB, an authentication request, such as an NTLM or NTLMv2 Challenge/Response or a Kerberos ticket, is sent from the client to the server.

At this stage, the SessionKey is not yet available.

```
Client -> Server: SMB: C session setup & X
SMB Flags2 contains 0xC807
Binary: 00000000 00000000 11001000 00000111
                               ^      ^  ^

SECURITY_SIGNATURE: Bit2 (set)
```

After the packet is sent by the client, the sequence number is incremented to 1, which is the expected sequence number for the response packet from the server.

4. The server processes the request and sends an SMB_COM_SESSION_SETUP_ANDX response to the client.

   It is possible that multiple roundtrips of SessionSetupAndX MAY be required to complete a given authentication. If STATUS_MORE_PROCESSING_REQUIRED is returned, you would return to the previous step and repeat. The below example demonstrates what happens when STATUS_SUCCESS is returned. Similarly, if this authentication was for Anonymous or Guest, signing would not be activated at this time.

```
Server -> Client: SMB: R session setup & X
SMB Flags2 contains 0xC807
Binary: 00000000 00000000 11001000 00000111
                                     ^  ^

SECURITY_SIGNATURE: Bit2 (set)
```

   The server sets the sequence number to 1 for the response packet, and generates the **SecuritySignature** as follows:

   The server places the sequence number (1) in the **SecuritySignature** field of the SMB header, and an MD5 hash is performed on the SessionKey + SMB packet. This results in a 16-byte value. The first 8 bytes of the computed hash (AB 44 C4 76 45 84 1A 6A) are placed in the **SecuritySignature** field and sent to the client.

```
00000:  00 11 43 02 26 E6 00 C0 4F 60 2E 45 08 00 45 00   ..C.&f.@O'.E..E.
00010:  01 78 85 60 40 00 80 32 F6 4B AC 1B 92 B9 AC 1B   .x&'@.,2vK,.9,.
00020:  92 B7 88 F2 96 BD 00 00 00 14 01 BD 05 48 8B A1   "Fr=.....=.H9!
00030:  8F 6C C1 3F C0 39 50 18 FF F0 84 70 00 00 00 00   lA?@9P.pp....
00040:  01 2F FF 53 4D 42 73 00 00 00 00 98 07 C8 00 00   ./SMBs....\.H..
00050: >AB 44 C4 76 45 84 1A 6A<00 00 00 00 FF FE 00 08   +DDvE.j....~..
00060:  40 00 04 FF 00 2F 01 00 00 A2 00 04 01 A1 81 9F   @.../..."...!x
00070:  30 81 9C A0 03 0A 01 00 A1 0B 06 09 2A 86 48 82   0S
....!...* H
```

After the server sends the packet, the sequence number is incremented to 2, which is the expected sequence number for the next SMB packet from the client.

5. The client processes the response and obtains the SessionKey.

```
SMB Flags2 contains 0xC807
Binary: 00000000 00000000 11001000 00000111
                                     ^  ^

SECURITY_SIGNATURE: Bit2 (set)
```

The expected sequence number is 1 for the response packet from the server.

The client saves the **SecuritySignature** in the response packet. The expected sequence number (1) is placed in the **SecuritySignature** field of the SMB header, and an MD5 hash is performed on the SessionKey SMB packet. This results in a 16-byte value. The first 8 bytes of the computed hash are compared with the one sent by the server (AB 44 C4 76 45 84 1A 6A) to validate the SMB packet.

6. The client proceeds further and sends an SMB_COM_TREE_CONNECT_ANDX request SMB.

```
Client -> Server: SMB: C tree connect & X, Share
```

The client sequence number is now incremented. The new value is 2.

The sequence number (2) is placed in the **SecuritySignature** field of the SMB header, and an MD5 hash is performed on the 16-byte SessionKey + SMB packet. This results in a 16-byte value. The first 8 bytes (in this case, A5 B0 43 DC 07 51 0F 8B) are placed in the **SecuritySignature** field in the SMB header and sent to the server.

```
00000:  00 C0 4F 60 2E 45 00 11 43 02 26 E6 08 00 45 00   .@O'.E..C.&f..E.
00010:  00 98 21 48 40 00 80 32 5B 44 AC 1B 92 B7 AC 1B   .\!H@.,2[D,.",.
00020:  92 B9 C4 70 3D 34 00 00 00 1C 05 48 01 BD C1 3F   9Dp=4.....H.=A?
00030:  C0 39 8B A1 90 9F 50 18 42 EF D0 D6 00 00 00 00   @99!xP.BoPV....
00040:  00 54 FF 53 4D 42 75 00 00 00 00 18 07 C8 00 00   .TSMBu......H..
00050: >A5 B0 43 DC 07 51 0F 8B<00 00 00 00 FF FE 00 08   %0C\.Q.9....~..
00060:  80 00 04 FF 00 54 00 0C 00 01 00 29 00 00 5C 00   ,...T.....)..\.
00070:  5C 00 4D 00 4F 00 48 00 41 00 4B 00 34 00 31 00   \.M.O.H.A.K.4.1.
```

The sequence continues until the session is terminated.

In the case where extended security is not used, the same process is followed. However, the MD5 Hash is performed on the 16-byte session key + NTLM challenge response + SMB packet with the appropriate sequence number. The NTLM challenge response is the authentication received in the SMB_COM_SESSION_SETUP_ANDX request in the **CaseSensitivePassword** field if NTLM was used for authentication, or in the **CaseInsensitivePassword** field if LM authentication was used.

4.4 Copy File (Remote to Local)

The following example illustrates the sequence of operations during copying of a file from a remote location to the local machine. The example assumes that the connection establishment and session management have already taken place.

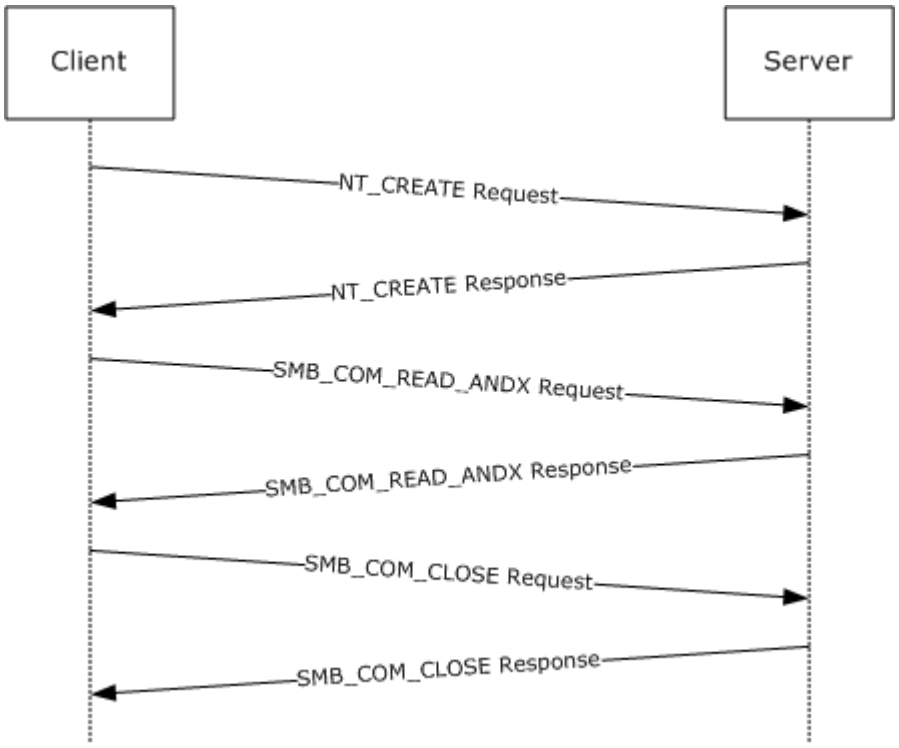


Figure 6: Copy file (remote to local) sequence

In the diagram above, the first frame is to open the remote file for read access. The subsequent frames read the data from the file, and then close the file. In between the read and the close, the data is written to the local file.

NT_CREATE_ANDX

```
Client -> Server: SMB: C NT Create Andx, Dialect = NTLM 0.12
SMB: Tree ID      (Tid) = 2049 (0x801)
SMB: Process ID   (Pid) = 3592 (0xE08)
SMB: User ID      (Uid) = 2048 (0x800)
SMB: Multiplex ID (Mid) = 1712 (0x6B0)
SMB: Command = C NT create & X
        SMB: Desired Access = 0x00020089
            SMB: ...............................1 = Read Data Allowed
            SMB: ..............................0. = Write Data Denied
            SMB: .............................0.. = Append Data Denied
            SMB: ............................1... = Read EA Allowed
            SMB: ...........................0.... = Write EA Denied
            SMB: ..........................0..... = File Execute Denied
            SMB: .........................0...... = File Delete Denied
            SMB: .......................1....... = File Read Attributes Allowed
            SMB: ......................0........ = File Write Attributes Denied
        SMB: NT File Attributes = 0x00000080
            SMB: ...............................0 = Not Read Only
```

```
            SMB: ..............................0. = Not Hidden
            SMB: ..............................0.. = Not System
            SMB: ............................0.... = Not Directory
            SMB: ..........................0..... = Not Archive
            SMB: ........................0...... = Not Device
            SMB: ......................1....... = Normal
            SMB: .....................0........ = Not Temporary
            SMB: ....................0......... = Not Sparse File
            SMB: ...................0.......... = Not Reparse Point
            SMB: ..................0........... = Not Compressed
            SMB: .................0............ = Not Offline
            SMB: ................0............. =
CONTENT INDEXED
            SMB: ................0............. = Not Encrypted
        SMB: File Share Access = 0x00000003
            SMB: ..............................1 = Read allowed
            SMB: .............................1. = Write allowed
            SMB: ............................0.. = Delete not
allowed
        SMB: Create Disposition = Open:  If exist, Open, else fail
        SMB: Create Options = 68 (0x44)
            SMB: ..............................0 = non-directory
            SMB: .............................0. = non-write through
            SMB: ............................1.. = Data must be written to the file
sequentially
            SMB: ............................0... = intermediate buffering allowed
            SMB: ...........................0.... = IO alerts bits not set
            SMB: ..........................0..... = IO non-alerts bit not set
            SMB: ........................1...... = Operation is on a non-directory file
            SMB: .......................0....... = tree connect bit not set
            SMB: ......................0........ = complete if oplocked bit is not set
            SMB: .....................0......... = no EA knowledge bit is not set
            SMB: ....................0.......... = 8.3 filenames bit is not set
            SMB: ...................0........... = random access bit is not set
            SMB: ..................0............ = delete on close bit is not set
            SMB: .................0............. = open by filename
            SMB: ................0.............. = open for backup bit not set
        SMB: File name =\filename.txt
```

NT_CREATE_ANDX Response

```
Client -> Server: SMB: C NT Create Andx, Dialect = NTLM 0.12
SMB: Tree ID      (Tid) = 2049 (0x801)
SMB: Process ID   (Pid) = 3592 (0xE08)
SMB: User ID      (Uid) = 2048 (0x800)
SMB: Multiplex ID (Mid) = 1712 (0x6B0)
SMB: Command = R NT create & X
        SMB: Oplock Level = Batch
        SMB: File ID (Fid) = 16389 (0x4005)

        SMB: NT File Attributes = 0x00000020
            SMB: ..............................0 = Not Read Only
            SMB: .............................0. = Not Hidden
            SMB: ............................0.. = Not System
            SMB: ...........................0.... = Not Directory
            SMB: ..........................1..... = Archive
            SMB: .........................0...... = Not Device
            SMB: ........................0....... = Not Normal
            SMB: .......................0........ = Not Temporary
            SMB: ......................0......... = Not Sparse File
```

```
SMB: .....................0.......... = Not Reparse Point
SMB: ....................0........... = Not Compressed
SMB: ...................0............ = Not Offline
SMB: ..................0............. = CONTENT_INDEXED
SMB: .................0.............. = Not Encrypted
```

SMB_COM_READ_ANDX Request

```
Client -> Server: SMB: C Read Andx, Dialect = NTLM 0.12
        SMB: Tree ID      (Tid) = 2049 (0x801)
        SMB: Process ID   (Pid) = 65279 (0xFEFF)
        SMB: User ID      (Uid) = 2048 (0x800)
        SMB: Multiplex ID (Mid) = 1744 (0x6D0)
SMB: Command = C read & X
        SMB: File ID (Fid) = 16389 (0x4005)
        SMB: Max count = 1596 (0x63C)
        SMB: Min count = 1596 (0x63C)
        SMB: Bytes left = 1596
```

SMB_COM_READ_ANDX Response

```
Client -> Server: SMB: R Read Andx, Dialect = NTLM 0.12
        SMB: Tree ID      (Tid) = 2049 (0x801)
        SMB: Process ID   (Pid) = 65279 (0xFEFF)
        SMB: User ID      (Uid) = 2048 (0x800)
        SMB: Multiplex ID (Mid) = 1744 (0x6D0)
SMB: Command = C read & X
        SMB: Data length = 1596 (0x63C)
        SMB: Data offset = 60 (0x3C)
        SMB: Byte count = 1597

Data = 00 90 27 D0 C4 6F 00 90 27 66 6D BE 08 00 45 00 ……
```

SMB_COM_CLOSE Request

```
Client -> Server: SMB: C Close, Dialect = NTLM 0.12
        SMB: Tree ID      (Tid) = 2049 (0x801)
        SMB: Process ID   (Pid) = 65279 (0xFEFF)
        SMB: User ID      (Uid) = 2048 (0x800)
        SMB: Multiplex ID (Mid) = 1984 (0x7C0)
SMB: Command = C Close
        SMB: File ID (Fid) = 16389 (0x4005)
```

SMB_COM_CLOSE Response

```
Client -> Server: SMB: R Close, Dialect = NTLM 0.12
        SMB: Tree ID      (Tid) = 2049 (0x801)
        SMB: Process ID   (Pid) = 65279 (0xFEFF)
```

```
        SMB: User ID      (Uid) = 2048 (0x800)
        SMB: Multiplex ID (Mid) = 1984 (0x7C0)
```

4.5 Copy File (Local to Remote)

The following example illustrates the sequence of operations while copying a local file to a remote share. The frames do not include the connection establishment or session management, for example.

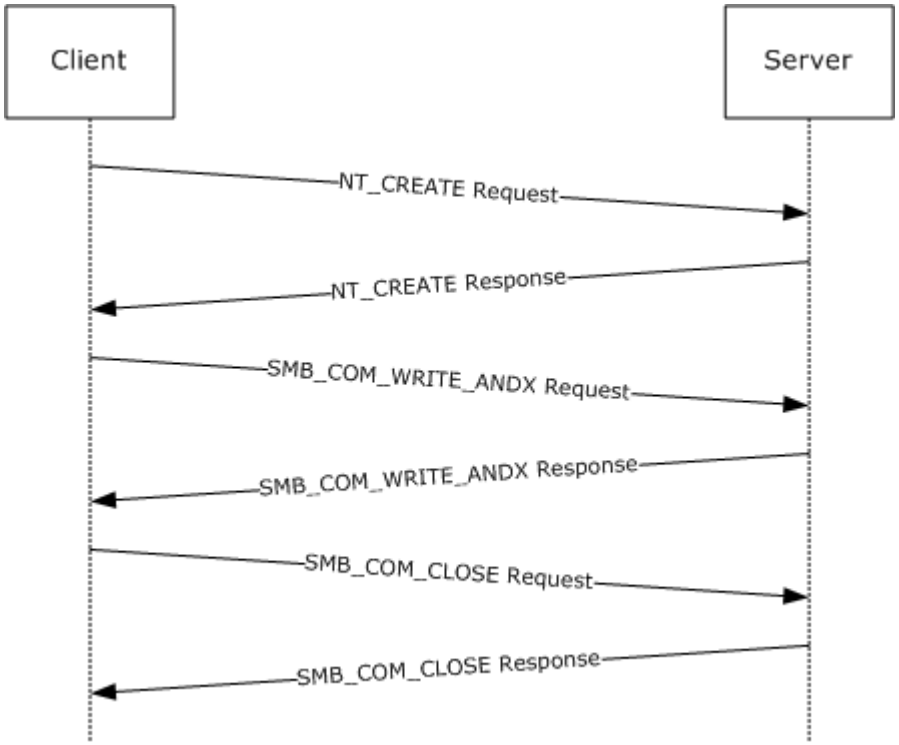


Figure 7: Copy file (local to remote) sequence

In the frames above, the remote file is first created with the SMB_COM_NT_CREATE_ANDX request. The data from the local file is then written to the remote file and subsequently the file is closed.

NT_CREATE_ANDX

```
Client -> Server: SMB: C NT Create Andx, Dialect = NTLM 0.12
        SMB: Tree ID      (Tid) = 2049 (0x801)
        SMB: Process ID   (Pid) = 3592 (0xE08)
        SMB: User ID      (Uid) = 2048 (0x800)
        SMB: Multiplex ID (Mid) = 2288 (0x8F0)
SMB: Command = C NT create & X
        SMB: Desired Access = 0x00030197
            SMB: .............................1 = Read Data Allowed
            SMB: .............................1. = Write Data Allowed
```

```
            SMB: ..............................1.. = Append Data Allowed
            SMB: ..............................0... = Read EA Denied
            SMB: ..............................1.... = Write EA Allowed
            SMB: ..............................0..... = File Execute Denied
            SMB: ..............................0...... = File Delete Denied
            SMB: ..............................1....... = File Read Attributes Allowed
            SMB: ..............................1........ = File Write Attributes Allowed
        SMB: NT File Attributes = 0x00000020
            SMB: ..............................0 = Not Read Only
            SMB: ..............................0. = Not Hidden
            SMB: ..............................0.. = Not System
            SMB: ..............................0.... = Not Directory
            SMB: ..............................1..... = Archive
            SMB: ..............................0...... = Not Device
            SMB: ..............................0....... = Not Normal
            SMB: ..............................0........ = Not Temporary
            SMB: ..............................0......... = Not Sparse File
            SMB: ..............................0.......... = Not Reparse Point
            SMB: ..............................0........... = Not Compressed
            SMB: ..............................0............ = Not Offline
            SMB: ..............................0............. =
CONTENT INDEXED
            SMB: ..............................0.............. = Not Encrypted
        SMB: File Share Access = 0x00000000
            SMB: ..............................0 = Read not allowed
            SMB: ..............................0. = Write not allowed
            SMB: ..............................0.. = Delete not allowed
        SMB: Create Disposition = Overwrite If:  If exist, open
and overwrite, else create it
        SMB: Create Options = 68 (0x44)
            SMB: ..............................0 = non-directory
            SMB: ..............................0. = non-write through
            SMB: ..............................1.. = Data must be written to the file
sequentially
            SMB: ..............................0... = intermediate buffering allowed
            SMB: ..............................0.... = IO alerts bits not set
            SMB: ..............................0..... = IO non-alerts bit not set
            SMB: ..............................1...... = Operation is on a non-directory file
            SMB: ..............................0....... = tree connect bit not set
            SMB: ..............................0........ = complete if oplocked bit is not set
            SMB: ..............................0......... = no EA knowledge bit is not set
            SMB: ..............................0.......... = 8.3 filenames bit is not set
            SMB: ..............................0........... = random access bit is not set
            SMB: ..............................0............ = delete on close bit is not set
            SMB: ..............................0............. = open by filename
            SMB: ..............................0.............. = open for backup bit not set
        SMB: File name =\filename.txt
```

NT_CREATE_ANDX Response

```
Client -> Server: SMB: R NT Create Andx, Dialect = NTLM 0.12
SMB: Tree ID      (Tid) = 2049 (0x801)
SMB: Process ID   (Pid) = 3592 (0xE08)
SMB: User ID      (Uid) = 2048 (0x800)
SMB: Multiplex ID (Mid) = 1712 (0x6B0)
SMB: Command = C NT create & X
        SMB: Oplock Level = Batch
        SMB: File ID (Fid) = 16392 (0x4008)

        SMB: NT File Attributes = 0x00000020
```

```
SMB: ..............................0 = Not Read Only
SMB: .............................0. = Not Hidden
SMB: ............................0.. = Not System
SMB: ...........................0.... = Not Directory
SMB: ..........................1..... = Archive
SMB: .........................0...... = Not Device
SMB: ........................0....... = Not Normal
SMB: .......................0........ = Not Temporary
SMB: ......................0......... = Not Sparse File
SMB: .....................0.......... = Not Reparse Point
SMB: ....................0........... = Not Compressed
SMB: ...................0............ = Not Offline
SMB: ..................0............. = CONTENT_INDEXED
SMB: .................0.............. = Not Encrypted
```

SMB_COM_WRITE_ANDX Request

```
Client -> Server: SMB: C Write Andx, Dialect = NTLM 0.12
        SMB: Tree ID      (Tid) = 2049 (0x801)
        SMB: Process ID   (Pid) = 65279 (0xFEFF)
        SMB: User ID      (Uid) = 2048 (0x800)
        SMB: Multiplex ID (Mid) = 2384 (0x950)
SMB: Command = C read & X
        SMB: File ID (Fid) = 16392 (0x4008)
        SMB: File offset = 0 (0x0)
        SMB: Data length = 1596 (0x63C)
Data = 00 90 27 66 6D BE 00 90 27 D0 C4 6F 08 00 45 00  ….
```

SMB_COM_READ_ANDX Response

```
Client -> Server: SMB: R Write Andx, Dialect = NTLM 0.12
        SMB: Tree ID      (Tid) = 2049 (0x801)
        SMB: Process ID   (Pid) = 65279 (0xFEFF)
        SMB: User ID      (Uid) = 2048 (0x800)
        SMB: Multiplex ID (Mid) = 2384 (0x950)
SMB: Command = C read & X
```

SMB_COM_CLOSE Request

```
Client -> Server: SMB: C Close, Dialect = NTLM 0.12
        SMB: Tree ID      (Tid) = 2049 (0x801)
        SMB: Process ID   (Pid) = 65279 (0xFEFF)
        SMB: User ID      (Uid) = 2048 (0x800)
        SMB: Multiplex ID (Mid) = 2400 (0x960)
SMB: Command = C Close
        SMB: File ID (Fid) = 16392 (0x4008)
```

SMB_COM_CLOSE Response

```
Client -> Server: SMB: R Close, Dialect = NTLM 0.12
        SMB: Tree ID      (Tid) = 2049 (0x801)
        SMB: Process ID   (Pid) = 65279 (0xFEFF)
        SMB: User ID      (Uid) = 2048 (0x800)
        SMB: Multiplex ID (Mid) = 2400 (0x960)
```

4.6 FSCTL_SRV_COPYCHUNK

The following example refers to the sequence of operations for a file copy in which the source and the destination are on the same server. The FSCTL_SRV_COPYCHUNK (section 2.2.14.7.3) is used. The following sequence assumes that the SMB connection to the server, SMB session establishment, and other operations have been completed.

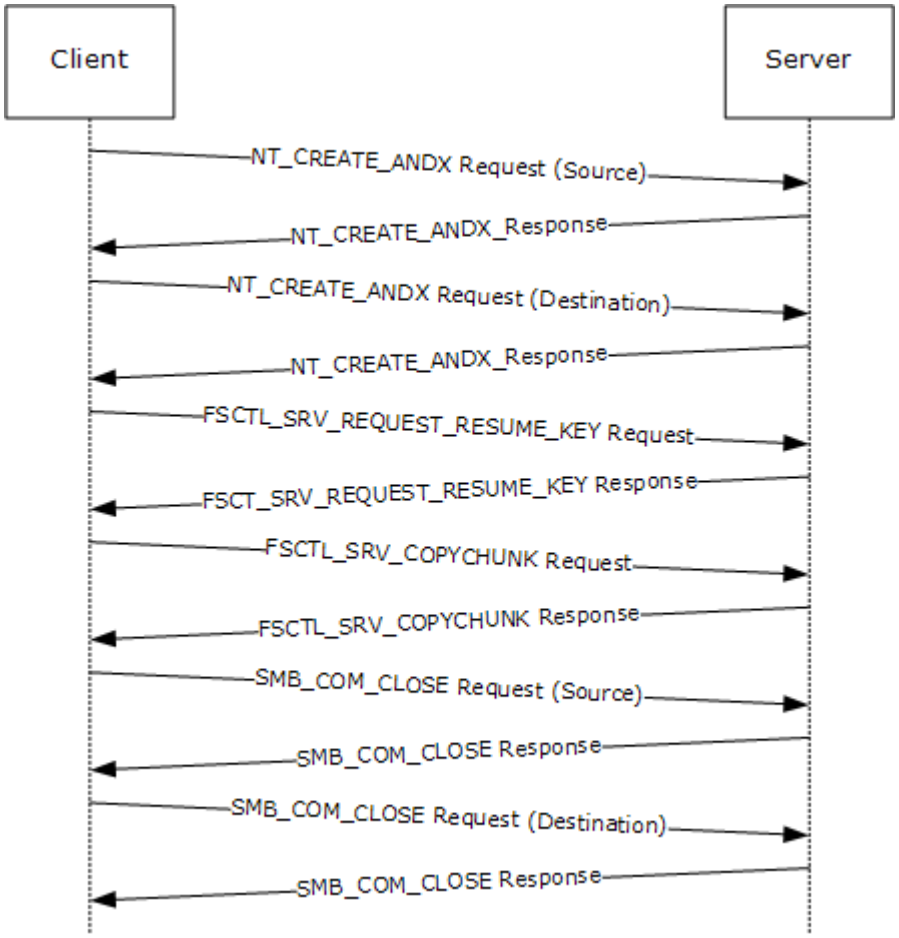


Figure 8: Copy file (from/to same remote server) sequence

The initial step in the sequence above is to open the source and the destination file using NT_CREATE_ANDX command. This is followed by the FSCTL_SRV_REQUEST_RESUME_KEY request (section 2.2.14.7.2). This is sent as an NT_TRANSACT_IOCTL with the file ID of the source file. The

server responds with the FSCTL_SRV_REQUEST_RESUME_KEY response (section 2.2.14.8.2). A 24-byte server copychunk resume key is returned.

NT_CREATE_ANDX Request (Source)

```
Client -> Server: SMB: C NT Create Andx, Dialect = NTLM 0.12
        SMB: Tree ID      (Tid) = 2049 (0x801)
        SMB: Process ID   (Pid) = 3592 (0xE08)
        SMB: User ID      (Uid) = 2048 (0x800)
        SMB: Multiplex ID (Mid) = 384 (0x180)
SMB: Command = C NT create & X
        SMB: Desired Access = 0x00020089
            SMB: ..............................1 = Read Data Allowed
            SMB: .............................0. = Write Data Denied
            SMB: ............................0.. = Append Data Denied
            SMB: ...........................1... = Read EA Allowed
            SMB: ..........................0.... = Write EA Denied
            SMB: .........................0..... = File Execute Denied
            SMB: ........................0...... = File Delete Denied
            SMB: .......................1....... = File Read Attributes Allowed
            SMB: ......................0........ = File Write Attributes Denied
        SMB: NT File Attributes = 0x00000000
            SMB: ..............................0 = Not Read Only
            SMB: .............................0. = Not Hidden
            SMB: ............................0.. = Not System
            SMB: ...........................0.... = Not Directory
            SMB: ..........................0..... = Not Archive
            SMB: .........................0...... = Not Device
            SMB: ........................0....... = Not Normal
            SMB: .......................0........ = Not Temporary
            SMB: ......................0......... = Not Sparse File
            SMB: .....................0.......... = Not Reparse Point
            SMB: ....................0........... = Not Compressed
            SMB: ...................0............ = Not Offline
            SMB: ..................0............. =
CONTENT INDEXED
            SMB: .................0.............. = Not Encrypted
        SMB: File Share Access = 0x00000005
            SMB: ..............................1 = Read allowed
            SMB: .............................0. = Write not allowed
            SMB: ............................1.. = Delete allowed
        SMB: Create Disposition = Open:  If exist, Open, else fail
        SMB: Create Options = 2097220 (0x200044)
            SMB: ..............................0 = non-directory
            SMB: .............................0. = non-write through
            SMB: ............................1.. = Data must be written to the
                                                   file sequentially
            SMB: ...........................0... = intermediate buffering allowed
            SMB: ..........................0.... = IO alerts bits not set
            SMB: .........................0..... = IO non-alerts bit not set
            SMB: ........................1...... = Operation is on a non-directory file
            SMB: .......................0....... = tree connect bit not set
            SMB: ......................0........ = complete if oplocked bit is not set
            SMB: .....................0......... = no EA knowledge bit is not set
            SMB: ....................0.......... = 8.3 filenames bit is not set
            SMB: ...................0........... = random access bit is not set
            SMB: ..................0............ = delete on close bit is not set
            SMB: .................0............. = open by filename
            SMB: ................0.............. = open for backup bit not set
        SMB: File name = sourcefile.txt
```

NT_CREATE_ANDX Response

```
Client -> Server: SMB: R NT Create Andx, Dialect = NTLM 0.12
        SMB: Tree ID      (Tid) = 2049 (0x801)
        SMB: Process ID   (Pid) = 3592 (0xE08)
        SMB: User ID      (Uid) = 2048 (0x800)
        SMB: Multiplex ID (Mid) = 384 (0x180)
SMB: Command = R NT create & X
        SMB: Oplock Level = II
        SMB: File ID (Fid) = 16386 (0x4002)

        SMB: NT File Attributes = 0x00000020
            SMB: ..............................0 = Not Read Only
            SMB: .............................0. = Not Hidden
            SMB: ............................0.. = Not System
            SMB: ...........................0.... = Not Directory
            SMB: ...........................1..... = Archive
            SMB: ..........................0...... = Not Device
            SMB: .........................0....... = Not Normal
            SMB: ........................0........ = Not Temporary
            SMB: .......................0......... = Not Sparse File
            SMB: ......................0.......... = Not Reparse Point
            SMB: .....................0........... = Not Compressed
            SMB: ....................0............ = Not Offline
            SMB: ...................0............. =
CONTENT_INDEXED
            SMB: .................0.............. = Not Encrypted
        SMB: File type = Disk file or directory
```

NT_CREATE_ANDX Request (Destination)

```
Client -> Server: SMB: C NT Create Andx, Dialect = NTLM 0.12
        SMB: Tree ID      (Tid) = 2049 (0x801)
        SMB: Process ID   (Pid) = 3592 (0xE08)
        SMB: User ID      (Uid) = 2048 (0x800)
        SMB: Multiplex ID (Mid) = 480 (0x1E0)
SMB: Command = C NT create & X
        SMB: Desired Access = 0x00030197
            SMB: ..............................1 = Read Data Allowed
            SMB: .............................1. = Write Data Allowed
            SMB: ............................1.. = Append Data Allowed
            SMB: ...........................0... = Read EA Denied
            SMB: ..........................1.... = Write EA Allowed
            SMB: .........................0..... = File Execute Denied
            SMB: ........................0...... = File Delete Denied
            SMB: .......................1....... = File Read Attributes Allowed
            SMB: ......................1........ = File Write Attributes Allowed
        SMB: NT File Attributes = 0x00000020
            SMB: ..............................0 = Not Read Only
            SMB: .............................0. = Not Hidden
            SMB: ............................0.. = Not System
            SMB: ...........................0.... = Not Directory
            SMB: ...........................1..... = Archive
            SMB: ..........................0...... = Not Device
            SMB: .........................0....... = Not Normal
            SMB: ........................0........ = Not Temporary
            SMB: .......................0......... = Not Sparse File
            SMB: ......................0.......... = Not Reparse Point
            SMB: .....................0........... = Not Compressed
            SMB: ....................0............ = Not Offline
            SMB: ...................0............. = CONTENT_INDEXED
```

```
            SMB: ..............0.............. = Not Encrypted
        SMB: File Share Access = 0x00000000
            SMB: ............................0 = Read not allowed
            SMB: ............................0. = Write not allowed
            SMB: ............................0.. = Delete not allowed
        SMB: Create Disposition = Overwrite If:  If exist, open and overwrite,
             else create it
        SMB: Create Options = 68 (0x44)
            SMB: ............................0 = non-directory
            SMB: ............................0. = non-write through
            SMB: ............................1.. = Data must be written to the file
sequentially
            SMB: ...........................0... = intermediate buffering allowed
            SMB: ...........................0.... = IO alerts bits not set
            SMB: ...........................0..... = IO non-alerts bit not set
            SMB: ...........................1...... = Operation is on a non-directory file
            SMB: ..........................0....... = tree connect bit not set
            SMB: ..........................0........ = complete if oplocked bit is not set
            SMB: ..........................0......... = no EA knowledge bit is not set
            SMB: ..........................0.......... = 8.3 filenames bit is not set
            SMB: ..........................0........... = random access bit is not set
            SMB: ..........................0............ = delete on close bit is not set
            SMB: ..........................0............. = open by filename
            SMB: ..........................0.............. = open for backup bit not set
        SMB: File name = destinationfile.txt
```

NT_CREATE_ANDX Response

```
Client -> Server: SMB: R NT Create Andx, Dialect = NTLM 0.12
        SMB: Tree ID      (Tid) = 2049 (0x801)
        SMB: Process ID   (Pid) = 3592 (0xE08)
        SMB: User ID      (Uid) = 2048 (0x800)
        SMB: Multiplex ID (Mid) = 480 (0x1E0)
SMB: Command = R NT create & X
        SMB: Oplock Level = Batch
        SMB: File ID (Fid) = 16387 (0x4003)

        SMB: NT File Attributes = 0x00000020
        SMB: ..............................0 = Not Read Only
        SMB: ..............................0. = Not Hidden
        SMB: ..............................0.. = Not System
        SMB: ............................0.... = Not Directory
        SMB: ............................1..... = Archive
        SMB: ...........................0...... = Not Device
        SMB: ...........................0....... = Not Normal
        SMB: ..........................0........ = Not Temporary
        SMB: ..........................0......... = Not Sparse File
        SMB: ..........................0.......... = Not Reparse Point
        SMB: ..........................0........... = Not Compressed
        SMB: ..........................0............ = Not Offline
        SMB: ..........................0............. = CONTENT_INDEXED
        SMB: ..........................0.............. = Not Encrypted
    SMB: File type = Disk file or directory
```

FSCTL_SRV_REQUEST_RESUME_KEY Request

```
Client -> Server: SMB: C NT Transact, Dialect = NTLM 0.12
NT IOCTL Function Code 0x00140078 FSCTL_SRV_REQUEST_RESUME_KEY
File ID (Fid) = 16386 (0x4002)
```

FSCTL_SRV_REQUEST_RESUME_KEY Response

```
Client -> Server: SMB: R NT Transact, Dialect = NTLM 0.12
NT IOCTL Function Code 0x00140078 FSCTL_SRV_REQUEST_RESUME_KEY
File ID (Fid) = 16386 (0x4002)
Key = 2D 0B 00 00 01 00 00 00 59 84 0C 62 1B 84 C6 01 08 0E 00 00 00 00 00 00
ContextLength = 0
```

This is followed by a FSCTL_SRV_COPYCHUNK request. The request uses the resume key generated above.

FSCTL_SRV_COPYCHUNK Request

```
Client -> Server: SMB: C NT Transact, Dialect = NTLM 0.12
NT IOCTL Function Code 0x001440F2 FSCTL SRV COPYCHUNK
File ID (Fid) = 16387 (0x4003)
Key = 2D 0B 00 00 01 00 00 00 59 84 0C 62 1B 84 C6 01 08 0E 00 00 00 00 00 00
ChunkCount = 1 (01 00 00 00)
Reserved = 0 (00 00 00 00)

List:
    SourceOffset = 0  (00 00 00 00 00 00 00 00)
    DestinationOffset = 0 (00 00 00 00 00 00 00 00)
    Length = 1731 (3C 06 00 00)
```

FSCTL_SRV_COPYCHUNK Response

```
Client -> Server: SMB: R NT Transact, Dialect = NTLM 0.12
NT IOCTL Function Code 0x001440F2 FSCTL_SRV_COPYCHUNK
File ID (Fid) = 16387 (0x4003)
ChunksWritten = 1 (01 00 00 00)
ChunkBytesWritten = 0 (00 00 00 00)
TotalBytesWritten = 1731 (3C 06 00 00)
```

The final step is to close the source and the destination file with SMB_COM_CLOSE commands.

SMB_COM_CLOSE Request (Source)

```
Client -> Server: SMB: C Close, Dialect = NTLM 0.12
        SMB: Tree ID      (Tid) = 2049 (0x801)
        SMB: Process ID   (Pid) = 65279 (0xFEFF)
        SMB: User ID      (Uid) = 2048 (0x800)
        SMB: Multiplex ID (Mid) = 640 (0x280)
SMB: Command = C Close
        SMB: File ID (Fid) = 16386 (0x4002)
```

SMB_COM_CLOSE Response

```
Client -> Server: SMB: R Close, Dialect = NTLM 0.12
        SMB: Tree ID      (Tid) = 2049 (0x801)
        SMB: Process ID   (Pid) = 65279 (0xFEFF)
        SMB: User ID      (Uid) = 2048 (0x800)
        SMB: Multiplex ID (Mid) = 640 (0x280)
```

SMB_COM_CLOSE Request (Destination)

```
Client -> Server: SMB: C Close, Dialect = NTLM 0.12
        SMB: Tree ID      (Tid) = 2049 (0x801)
        SMB: Process ID   (Pid) = 65279 (0xFEFF)
        SMB: User ID      (Uid) = 2048 (0x800)
        SMB: Multiplex ID (Mid) = 656 (0x290)
SMB: Command = C Close
        SMB: File ID (Fid) = 16387 (0x4003)
```

SMB_COM_CLOSE Response

```
Client -> Server: SMB: R Close, Dialect = NTLM 0.12
        SMB: Tree ID      (Tid) = 2049 (0x801)
        SMB: Process ID   (Pid) = 65279 (0xFEFF)
        SMB: User ID      (Uid) = 2048 (0x800)
        SMB: Multiplex ID (Mid) = 656 (0x290)
```

4.7 TRANS_TRANSACT_NMPIPE

The following example illustrates how the TRANS_TRANSACT_NMPIPE is used.

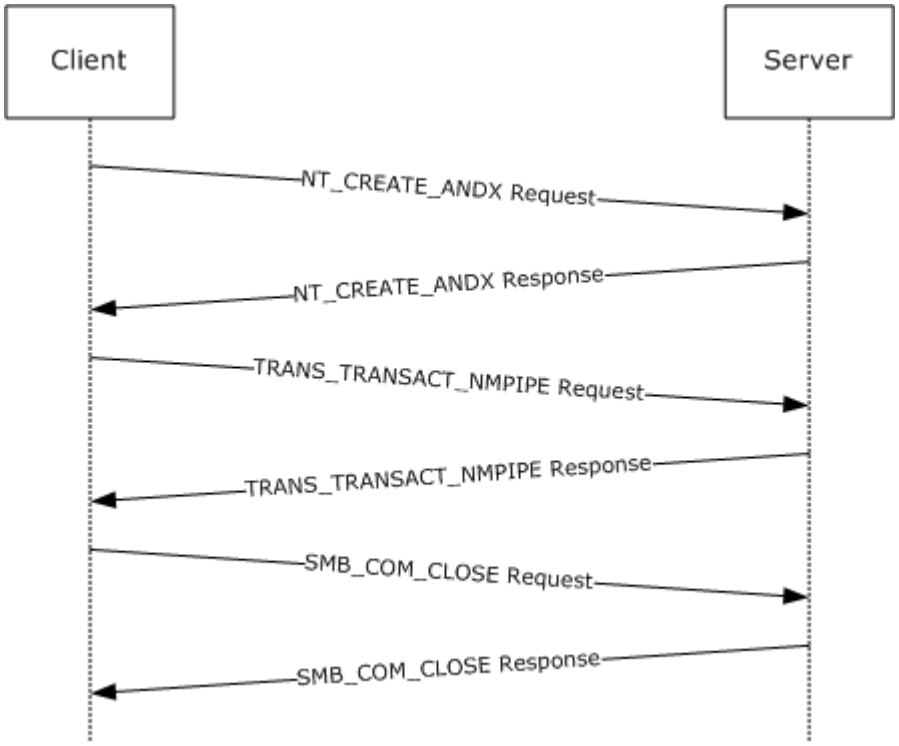


Figure 9: Named pipe request sequence

The first frame contains the NT_CREATE request to the named pipe. The TRANS_TRANSACT_NMPIPE is then issued against the file ID assigned in the NT_CREATE_ANDX response.

NT_CREATE_ANDX

```
Client -> Server: SMB: C NT Create Andx, Dialect = NTLM 0.12
        SMB: Tree ID      (Tid) = 2048 (0x800)
        SMB: Process ID   (Pid) = 2292 (0x8F4)
        SMB: User ID      (Uid) = 2048 (0x800)
        SMB: Multiplex ID (Mid) = 4048 (0xFD0)
SMB: Command = C NT create & X
        SMB: Desired Access = 0x0002019F
            SMB: .............................1 = Read Data Allowed
            SMB: ............................1. = Write Data Allowed
            SMB: ...........................1.. = Append Data Allowed
            SMB: ..........................1... = Read EA Allowed
            SMB: .........................1.... = Write EA Allowed
            SMB: ........................0..... = File Execute Denied
            SMB: .......................0...... = File Delete Denied
            SMB: ......................1....... = File Read Attributes Allowed
            SMB: .....................1........ = File Write Attributes Allowed
        SMB: NT File Attributes = 0x00000000
            SMB: .............................0 = Not Read Only
            SMB: ............................0. = Not Hidden
            SMB: ...........................0.. = Not System
            SMB: ..........................0.... = Not Directory
            SMB: .........................0..... = Not Archive
            SMB: ........................0...... = Not Device
            SMB: .......................0....... = Not Normal
            SMB: ......................0........ = Not Temporary
            SMB: .....................0......... = Not Sparse File
```

```
            SMB: ......................0.......... = Not Reparse Point
            SMB: .....................0........... = Not Compressed
            SMB: ....................0............ = Not Offline
            SMB: ...................0............. =
CONTENT INDEXED
            SMB: ..................0.............. = Not Encrypted
        SMB: File Share Access = 0x00000003
            SMB: ..............................1 = Read allowed
            SMB: .............................1. = Write allowed
            SMB: ............................0.. = Delete not
allowed

        SMB: Create Disposition = Open:  If exist, Open, else fail
        SMB: Create Options = 4194368 (0x400040)
            SMB: ..............................0 = non-directory
            SMB: .............................0. = non-write through
            SMB: ............................0.. = non-sequential writing allowed
            SMB: ...........................0... = intermediate buffering allowed
            SMB: ..........................0.... = IO alerts bits not set
            SMB: .........................0..... = IO non-alerts bit not set
            SMB: ........................1...... = Operation is on a non-directory file
            SMB: .......................0....... = tree connect bit not set
            SMB: ......................0........ = complete if oplocked bit is not set
            SMB: .....................0......... = no EA knowledge bit is not set
            SMB: ....................0.......... = 8.3 filenames bit is not set
            SMB: ...................0........... = random access bit is not set
            SMB: ..................0............ = delete on close bit is not set
            SMB: .................0............. = open by filename
            SMB: ................0.............. = open for backup bit not set

        SMB: File name =\srvsvc
```

NT_CREATE_ANDX Response

```
Client -> Server: SMB: R NT Create Andx, Dialect = NTLM 0.12
        SMB: Tree ID      (Tid) = 2048 (0x800)
        SMB: Process ID   (Pid) = 2292 (0x8F4)
        SMB: User ID      (Uid) = 2048 (0x800)
        SMB: Multiplex ID (Mid) = 4048 (0xFD0)
SMB: Command = R NT create & X
        SMB: Oplock Level = NONE
        SMB: File ID (Fid) = 16385 (0x4001)

        SMB: NT File Attributes = 0x00000080
            SMB: ..............................0 = Not Read Only
            SMB: .............................0. = Not Hidden
            SMB: ............................0.. = Not System
            SMB: ...........................0.... = Not Directory
            SMB: ..........................0..... = Not Archive
            SMB: .........................0...... = Not Device
            SMB: ........................1....... = Normal
            SMB: .......................0........ = Not Temporary
            SMB: ......................0......... = Not Sparse File
            SMB: .....................0.......... = Not Reparse Point
            SMB: ....................0........... = Not Compressed
            SMB: ...................0............ = Not Offline
            SMB: ..................0............. = CONTENT_INDEXED
            SMB: .................0.............. = Not Encrypted
        SMB: File type = Message mode named pipe
```

SMB_COM_TRANSACTION Request

```
Client -> Server: SMB: C transact TransactNmPipe, Dialect = NTLM
0.12
        SMB: Tree ID      (Tid) = 2048 (0x800)
        SMB: Process ID   (Pid) = 2292 (0x8F4)
        SMB: User ID      (Uid) = 2048 (0x800)
        SMB: Multiplex ID (Mid) = 4096 (0x1000)
SMB: Command = C transact
        SMB: Data bytes = 76 (0x4C)
        SMB: Data offset = 84 (0x54)
        SMB: Setup words
        SMB: Pipe function = Transact named pipe (TransactNmPipe)
        SMB: File ID (Fid) = 16385 (0x4001)
Data = 00 90 27 66 6D BE 00 90 27 D0 C4 6F 08 00 45 00   ……
```

SMB_COM_TRANSACTION Response

```
Client -> Server: SMB: R transact TransactNmPipe, Dialect = NTLM
0.12
        SMB: Tree ID      (Tid) = 2048 (0x800)
        SMB: Process ID   (Pid) = 2292 (0x8F4)
        SMB: User ID      (Uid) = 2048 (0x800)
        SMB: Multiplex ID (Mid) = 4096 (0x1000)
SMB: Command = R transact
        SMB: Data bytes = 120 (0x78)
        SMB: Data offset = 56 (0x38)

DATA = 00 90 27 D0 C4 6F 00 90 27 66 6D BE 08 00 45 00 ….
```

SMB_COM_CLOSE Request

```
Client -> Server: SMB: C Close, Dialect = NTLM 0.12
SMB: Tree ID      (Tid) = 2048 (0x800)
        SMB: Process ID   (Pid) = 65279 (0xFEFF)
        SMB: User ID      (Uid) = 2048 (0x800)
        SMB: Multiplex ID (Mid) = 4112 (0x1010)
SMB: Command = C Close
        SMB: File ID (Fid) = 16385 (0x4001)
```

SMB_COM_CLOSE Response

```
Client -> Server: SMB: R Close, Dialect = NTLM 0.12
  SMB: Tree ID      (Tid) = 2048 (0x800)
  SMB: Process ID   (Pid) = 65279 (0xFEFF)
  SMB: User ID      (Uid) = 2048 (0x800)
```

```
SMB: Multiplex ID (Mid) = 4112 (0x1010)
```

5 Security Considerations

The following sections specify security considerations for implementers of the SMB Protocol.

5.1 Security Considerations for Implementers

The CIFS Protocol contains support for NTLM but lacks message signing and support for new authentication protocols. The extensions defined in this document offer support for increased security in remote file and printer access via SMB.

The new previous versions support does potentially allow access to versions of a file that have been deleted or modified, so this can allow access to information that was not available without these extensions. However, this access is still subject to the same access checks that it is normally subject to.

The protocol does not sign oplock break requests from the server to the client if message signing is enabled. This can allow an attacker to affect performance, but does not allow an attacker to deny access or alter data.

The algorithm used for message signing has been shown to be subject to collision attacks. For more information, see [MD5Collision].

5.2 Index of Security Parameters

In addition to the NTLM challenge/response authentication support, as specified in [CIFS] section 2.10, these extensions enable support for Kerberos or any other protocol that can be encapsulated inside the extensible authentication package, as specified in [RFC4178].

Message signing uses the MD5 algorithm to hash the messages.

Extended message signing uses the HMAC_MD5 algorithm, as specified in [RFC2104], to alter the user's session key.

6 Appendix A: Windows Behavior

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

- Windows 95
- Windows 98
- Windows Me
- Windows NT
- Windows 2000
- Windows XP
- Windows Vista
- Windows Server 2003
- Windows Server 2008

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.

Note The entries in the following list that refer to Windows Server 2003 also represent the Windows behavior on Windows Server 2003 R2.

<1> Section 2.1.1: Windows-based clients and servers use TCP port 445 as the destination TCP port on the SMB server, the well-known port number assigned by IANA to Microsoft-DS.

<2> Section 2.1.1: The Direct TCP Transport MAY be used by Windows-based SMB clients and servers that run on Windows Server 2003, Windows Vista, Windows XP, and Windows 2000.

<3> Section 2.1.2: The NetBIOS Over IPX Transport MAY be used by Windows-based SMB clients and servers on Windows NT 4.0, Windows XP, Windows 2000, Windows Me, Windows 98, and Windows 95.

<4> Section 2.1.3: This field is normally set to 0 on the acknowledgment sent in the response to a session initialize frame.

<5> Section 2.1.3: This field is normally set to 0 on the session initialize frame sent by the initiator.

<6> Section 2.1.3: The NetBEUI transport MAY be used by Windows-based SMB clients and servers on Windows NT 4.0, Windows NT 3.51, Windows NT 3.5, Windows NT 3.1, Windows 2000, Windows Me, Windows 98, and Windows 95.

<7> Section 2.2: Windows-based clients and servers set fields and bit fields marked as Unused to 0 when being sent, and ignore these fields and bit fields when received.

<8> Section 2.2: Unless specifically required to return data, as specified for named pipe read operations and certain I/O control code requests, Windows-based SMB servers choose to return only the SMB header packet on error. Unless otherwise specified, a Windows-based SMB client expects that a server returns only the SMB header packet on error.

<9> Section 2.2.1: The **SMB_FLAGS_SEND_NO_ACK** bit field is not set when using direct TCP, NetBIOS over TCP, NetBIOS over IPX, or NetBEUI.

<10> Section 2.2.1: Windows-based clients and servers ignore this bit on other messages.

<11> Section 2.2.1: Windows-based SMB clients and servers support all of the **Flags2** bit fields, as specified above, when NT LM 0.12 or later is negotiated for the SMB dialect. Windows-based SMB clients and servers support the following **Flags2** bit fields when LM1.2X002 is negotiated for the SMB dialects: SMB_FLAGS2_KNOWS_EAS, SMB_FLAGS2_KNOWS_LONG_NAMES, and SMB_FLAGS2_PAGING.

<12> Section 2.2.1: Windows-based clients set this bit on some client requests. Windows-based servers ignore this bit when set on client requests.

<13> Section 2.2.1: Windows-based SMB clients on Windows 2000 and later versions support 32-bit process IDs and use this field when sending the following SMB messages: SMB_COM_NT_CREATE_AND and SMB_COM_OPEN_PRINT_FILE. Windows-based SMB clients on Windows 2000 and later versions also support and use this field when sending SMB_COM_NT_TRANSACT, SMB_COM_TRANSACTION, and SMB_COM_TRANSACTION2 messages when the server supports the CAP_NT_SMBS bit. (The CAP_NT_SMBS bit was set in the **Capabilities** field in the SMB_COM_NEGOTIATE (section 2.2.2) response.) Other Windows-based SMB clients do not support 32-bit process IDs and set this field to 0 when sending SMB messages. Windows-based SMB servers support 32-bit process IDs when receiving SMB messages.

<14> Section 2.2.2: Windows-based SMB clients send only a single SMB_COM_NEGOTIATE request over an SMB transport while an SMB connection is active on the SMB transport, as specified in [CIFS] section 4.1.1.

<15> Section 2.2.2: The list of all possible dialects recognized by various Windows-based clients and servers includes the following:

Symbolic name	DialectName string
PCNET1	PC NETWORK PROGRAM 1.0
XENIXCORE	XENIX CORE
PCLAN1	PCLAN1.0
MSNET103	MICROSOFT NETWORKS 1.03
MSNET30	MICROSOFT NETWORKS 3.0
LANMAN10	LANMAN1.0
WFW10	Windows for Workgroups 3.1a
DOSLANMAN12	DOS LM1.2X002
LANMAN12	LM1.2X002
DOSLANMAN21	DOS LANMAN2.1
LANMAN21	LANMAN2.1
NTLANMAN	NT LM 0.12
SMB2	SMB 2.001

Windows-based clients list SMB dialects in the **DialectName** field from the earliest supported dialect to the latest supported dialect. SMB clients that run on Windows 2000 and later send a **DialectName** array field that contains dialect strings for the following symbolic dialect names: PCNET1, LANMAN10, WFW10, LANMAN12, LANMAN21, and NTLANMAN. SMB clients that run on Windows NT 4.0 send a **DialectName** array field that contains dialect strings for the following symbolic dialect names: PCNET1, XENIXCORE, MSNET103, LANMAN10, WFW10, LANMAN12, LANMAN21, and NTLANMAN. SMB clients that run on Windows Me and Windows 98 send a **DialectName** array field that contains the dialect strings for the following symbolic dialect names: PCNET1, MSNET30, DOSLANMAN12, DOSLANMAN21, WFW10, and NTLANMAN. Windows Vista and future SMB clients that support SMB2 will also send the SMB2 dialect string, in addition to the dialect string sent by Windows 2000 and later as described above. If the SMB2 dialect is negotiated, the Server Message Block (SMB) Version 2.0 Protocol, as specified in [MS-SMB2], is used instead of [CIFS] with these extensions.

<16> Section 2.2.3: Windows-based SMB servers return STATUS_INVALID_SMB in the response and disconnect the underlying transport connection if more than a single SMB_COM_NEGOTIATE client request is received over the same SMB transport while an SMB connection is already established on the SMB transport.

<17> Section 2.2.3: Windows-based SMB servers set this field to 0x01 but do not enforce this limit, which allows an SMB client to establish more virtual circuits than allowed by this value. Because this limit is not enforced on Windows, SMB clients MAY ignore this limit and attempt to establish more than the number of virtual circuits allowed by this value. The Windows behavior of the SMB server allows a client to exceed this limit, but other server implementations can enforce this limit and not allow this to occur.

<18> Section 2.2.3: Windows defaults to a **MaxBufferSize** value of 16,644 bytes on server versions of Windows (for example, Windows Server 2003, Windows 2000 Server, and Windows NT Server). Windows defaults to a **MaxBufferSize** value of 4,356 bytes on client versions of Windows (for example, Windows XP, Windows 2000 Professional, Windows NT Workstation, and Windows 98).

<19> Section 2.2.3: Windows defaults to a **MaxRawSize** value of 65,536 bytes.

<20> Section 2.2.3: Windows-based clients and servers take advantage of CAP_INFOLEVEL_PASSTHRU, when available, to prevent the need to map from native file and directory information structures to comparable SMB structures. The CAP_INFOLEVEL_PASSTHRU capability bit was called CAP_QUADWORD_ALIGNED in some prerelease versions of Windows. The CAP_QUADWORD_ALIGNED capability was to indicate that Windows directory InformationLevel responses were quadword aligned. The CAP_QUADWORD_ALIGNED capability bit was never used in released Windows-based clients or servers, and is used for the CAP_INFOLEVEL_PASSTHRU bit.

In Windows 2000 and later, all clients and servers support CAP_LARGE_READX and CAP_LARGE_WRITEX, which permit file transfers larger than the negotiated **MaxBufferSize**.

Windows-based clients and servers do not support CAP_UNIX; therefore, this capability is never set.

Windows-based clients and servers do not support CAP_PERSISTENT_HANDLES; therefore, this capability is never set.

Windows servers do not set the CAP_DYNAMIC_REAUTH bit in the negotiate response, even if it supports dynamic reauthentication.

Windows servers do not set the CAP_DYNAMIC_REAUTH flag even if it supports dynamic re-authentication. On Windows XP and later, all clients and servers support dynamic reauthentication. The CAP_DYNAMIC_REAUTH capability bit was called CAP_BULK_TRANSFER in some early documentation on SMB from Microsoft and other vendors. The CAP_BULK_TRANSFER capability was

supposed to indicate if the server supported SMB_COM_READ_BULK and SMB_COM_WRITE_BULK commands, which were never implemented. The CAP_BULK_TRANSFER capability bit was never used in Windows-based clients or servers, and is used for the CAP_DYNAMIC_REAUTH bit.

The CAP_COMPRESSED_DATA capability bit was to indicate if a server supported compressed SMB packets. This feature was never specified, implemented, or used. Windows-based clients and servers do not support CAP_COMPRESSED_DATA, and so this capability will never be set.

<21> Section 2.2.3: Windows 2000 and later clients use the ServerGuid in the manner defined above.

<22> Section 2.2.3.1: Windows-based SMB servers set this field to one but do not enforce this limit, which allows an SMB client to establish more virtual circuits than allowed by this value. Because this limit is not enforced on Windows, SMB clients MAY ignore this limit and attempt to establish more than the number of virtual circuits allowed by this value. The Windows behavior of the SMB server allows a client to exceed this limit, but other server implementations can enforce this limit and not allow this to occur.

<23> Section 2.2.3.1: Windows defaults to a **MaxBufferSize** value of 16,644 bytes on server versions of Windows (for example, Windows Server 2003, Windows 2000 Server, and Windows NT Server). Windows defaults to a **MaxBufferSize** value of 4,356 bytes on client versions of Windows (for example, Windows XP, Windows 2000 Professional, Windows NT Workstation, and Windows 98).

<24> Section 2.2.3.1: Windows defaults to a **MaxRawSize** value of 65,536 bytes.

<25> Section 2.2.3.1: Windows-based clients and servers take advantage of CAP_INFOLEVEL_PASSTHRU, when available, to prevent the need to map from native file and directory information structures to comparable SMB structures. The CAP_INFOLEVEL_PASSTHRU capability bit was called CAP_QUADWORD_ALIGNED in some prerelease versions of Windows. The CAP_QUADWORD_ALIGNED capability was to indicate that Windows directory InformationLevel responses were quadword aligned. The CAP_QUADWORD_ALIGNED capability bit was never used in released Windows-based clients or servers, and is used for the CAP_INFOLEVEL_PASSTHRU bit.

In Windows 2000 and later, all clients and servers support CAP_LARGE_READX and CAP_LARGE_WRITEX, which permit file transfers larger than the negotiated **MaxBufferSize**.

Windows-based clients and servers do not support CAP_UNIX, so this capability is never set.

Windows-based clients and servers do not support CAP_PERSISTENT_HANDLES, so this capability is never set.

Windows servers do not set the CAP_DYNAMIC_REAUTH flag even if they support dynamic re-authentication. On Windows XP and later, all clients and servers support dynamic reauthentication. The CAP_DYNAMIC_REAUTH capability bit was called CAP_BULK_TRANSFER in some early documentation on SMB from Microsoft and other vendors. The CAP_BULK_TRANSFER capability was supposed to indicate if the server supported SMB_COM_READ_BULK and SMB_COM_WRITE_BULK commands, which were never implemented. The CAP_BULK_TRANSFER capability bit was never used in Windows-based clients or servers, and is used for the CAP_DYNAMIC_REAUTH bit.

The CAP_COMPRESSED_DATA capability bit was to indicate if a server supports compressed SMB packets. This feature was never specified, implemented, or used. Windows-based clients and servers do not support CAP_COMPRESSED_DATA, so this capability is never set.

<26> Section 2.2.3.1: Windows clients and servers expect 8-byte encryption keys.

<27> Section 2.2.4: Windows defaults to a **MaxBufferSize** value of 16,644 bytes on server versions of Windows (for example, Windows Server 2003, Windows 2000 Server, and Windows NT

Server). Windows defaults to a **MaxBufferSize** value of 4,356 bytes on client versions of Windows (for example, Windows XP, Windows 2000 Professional, Windows NT Workstation, and Windows 98). Windows SMB clients and servers use the minimum value of the **MaxBufferSize** field in the SMB_COM_NEGOTIATE response (section 2.2.2) and the **MaxBufferSize** field in the SMB_COM_SESSION_SETUP_ANDX request (section 2.2.4) to set the size of the maximum buffer used for sending and receiving SMB messages for both the client and server.

<28> Section 2.2.4: Windows-based SMB servers set a limit for the **MaxNumberVcs** field in the SMB_COM_NEGOTIATE response (section 2.2.3) to 0x01, but do not enforce this limit. This allows an SMB client to establish more virtual circuits than allowed by the **MaxNumberVcs** field value. Because this limit is not enforced on Windows, SMB clients MAY ignore this limit and attempt to establish more than the number of virtual circuits allowed by this value. The Windows behavior of the SMB server allows a client to exceed this limit, but other server implementations MAY enforce this limit and not allow this to occur.

<29> Section 2.2.4: Windows 2000 and later servers ignore this field.

<30> Section 2.2.4: Windows XP and later clients set the CAP_DYNAMIC_REAUTH capability bit to indicate to the server that the client supports re-authentication when the Kerberos ticket for the session expires. Windows XP and later servers always perform re-authentication when the Kerberos ticket for the session expires, irrespective of whether the client has set the CAP_DYNAMIC_REAUTH flag or not.

<31> Section 2.2.4: SMB clients set this field based on the version and service pack level of the Windows operating system. A list of possible values for this field includes the following. Windows Vista sets this field to an empty string.

Windows OS version	NativeOS string
Windows Server 2003 SP1	Windows Server 2003 3790 Service Pack 1
Windows XP SP2	Windows 2002 Service Pack 2
Windows 2000	Windows 5.0
Windows NT 4.0	Windows NT 1381
Windows 98 Second Edition	Windows 4.0

<32> Section 2.2.4: Windows-based SMB clients set this field based on the version of the Windows operating system. A list of possible values for this field includes the following. Windows Vista sets this field to an empty string.

Windows OS version	NativeLANMan string
Windows Server 2003	Windows Server 2003 5.2
Windows XP SP2	Windows 2002 5.1
Windows 2000	Windows 2000 LAN Manager
Windows NT 4.0	(a NULL string)
Windows 98	Windows 4.0

<33> Section 2.2.5: Windows-based SMB servers set this field based on the version and service pack level of the Windows operating system. A list of possible values for this field includes the following:

Windows OS version	NativeOS string
Windows Server 2003 SP1	Windows Server 2003 3790 Service Pack 1
Windows XP SP2	Windows 2002 Service Pack 2
Windows 2000	Windows 5.0
Windows NT 4.0	Windows NT 1381
Windows 98 Second Edition	Windows 4.0

<34> Section 2.2.5: Windows-based SMB servers set this field based on the version of the Windows operating system. A list of possible values for this field includes the following:

Windows OS Version	NativeLANMan String
Windows Server 2003	Windows Server 2003 5.2
Windows XP SP2	Windows 2002 5.1
Windows 2000	Windows 2000 LAN Manager
Windows NT 4.0	(a NULL string)
Windows 98	Windows 4.0

<35> Section 2.2.6: Windows-based SMB clients and SMB servers use some special share names to represent some resources. Note that in the following table, the **Service** field that is equal to the special string of "?????" (five question marks) matches any type of device or resource. These share names are case insensitive and are as follows:

Path	Service	Description
\\SERVER\IPC$	IPC or ?????	Used for access to named pipes for use by remote APIs.
\\SERVER\C$	A: or ?????	Hidden share created for each local drive on a Windows computer.
\\SERVER\PRINT$	A: or ?????	Share to host downloadable drivers for shared printers installed on the Windows computer.
\\SERVER\ADMIN$	A: or ?????	Share to the Windows system directory that is used for remote management or policy.
\\SERVER\NETLOGON	A: or ?????	Share on a Windows domain controller to share out policy files and other domain-specific information.
\\SERVER\SYSVOL	A: or ?????	Share on a Windows domain controller to share out policy files and other domain-specific information.
\\SERVER\printername	LPT1: or	Shared printer where printername is the name given to a shared

Path	Service	Description
	?????	printer on a Windows computer.
Printers and Faxes	LPT1: or ?????	Share that shows installed shared printers and faxes on a Windows computer.
Scheduled Tasks	?????	Share that shows tasks that are scheduled to run automatically on a Windows computer.

The only share that is required by a Windows-based client is the \\SERVER\IPC$ share. IPC$ is used for named pipe access, so opening the file \\SERVER\IPC$\sample is actually a request to open the named pipe \sample on the \\SERVER computer. Remote APIs over named pipes are supported by the SMB Protocol, so the IPC$ share is used by existing Windows services to execute remote API calls on a remote machine.

There is one additional use of the \\SERVER\IPC$ share by Windows-based clients. Since IPC$ is the only share guaranteed to exist on a server, a SMB_COM_TREE_CONNECT_ANDX client request to this share can return information on additional features supported by the server and shares on that server. The SMB_COM_TREE_CONNECT_ANDX response includes an **OptionalSupport** field with bit fields that indicate support for specific features. The range of features include support for search bits (filtering directory searches based on file attributes), enabling Distributed File System (DFS) shares on the server, client-side caching options, server permission to cache namespace attributes, and signing key protection.

Any share whose textual name ends in a $ is classified as a hidden share. These shares are returned in share enumeration calls (for more information about NetrShareEnum, see MS-SRVS section 3.1.4.8), but the Windows user interface implementation does not display these shares to the user. This MAY not be the case for applications from other vendors.

There are other hidden shares that a server MAY optionally create. Existing Windows-based clients and servers normally create a hidden share for each of the local drives existing on the machine, naming them after the drive names. For example, a \\SERVER\C$ share would be created for the C: drive and a similar \\SERVER\D$ share would be created for the D: drive. On a Windows-based computer with NTFS used for the local file system, these shares have an access control list (ACL) applied such that only administrators have access. These C$ shares are optional, and there is nothing specific to the SMB Protocol that requires them.

Windows-based computers often create a PRINT$ share to host downloadable drivers for shared printers installed on the machine. This share would commonly point to C:\WINDOWS\System32\spool\drivers on recent versions of Windows. The use and management of this share is handled by the Windows spooler service on this computer, which MAY refer clients to download drivers from this share. This share is optional, and there is nothing specific to the SMB Protocol that requires it.

A Printers and Faxes share is sometimes created to share a list of printers and faxes on a computer (limited to Windows XP and later). There are separate \\SERVER\printername shares created for each printer shared by a computer. The printer name used in the share name is the name assigned to the printer by Windows when the printer was shared as a resource. These shares are optional, and there is nothing specific to the SMB Protocol that requires them.

There are other shares that components running on a Windows NT-based computer MAY optionally create. Windows-based computers MAY create a \\SERVER\ADMIN$ share that shares out the system directory for remote management or policy use. This share would default to C:\WINDOWS on recent versions of Windows, but the name and location of the system directory is configurable during Windows installation. A Scheduled Tasks share is sometimes created to shows tasks that are

scheduled to run automatically on a computer. These shares are optional, and there is nothing specific to the SMB Protocol that requires them.

The NETLOGON and SYSVOL shares are used on Windows domain controllers to share out policy files and other domain-specific information.

These other shares are not required by any SMB server to fully interoperate with other client and servers in the role of a file server. The only share that is required is the \\SERVER|IPC$ as mentioned above, which is used for executing remote APIs.

<36> Section 2.2.7: SMB clients on Windows XP and Windows Vista client versions of Windows will cache directory information if this bit is set on a share. Earlier client versions will not cache the directory information. SMB clients on all server versions of Windows will not cache directory information by default if this bit is set on a share. Caching directory information by SMB clients on Windows 2000 Server and later server versions of Windows may be enabled via a Windows registry setting.

<37> Section 2.2.7: Windows Server 2003 and Windows Server 2008 will set this bit if short name generation is disabled in the file system which reduces aliasing, or if an administrator has explicitly enabled it.

<38> Section 2.2.7: Windows supports the notion of a guest account and sets this field to the access allowed for the guest account.

<39> Section 2.2.8: Windows-based SMB servers allow only a combination of the following shared access types on a named pipe: FILE_SHARE_READ and FILE_SHARE_WRITE. Windows-based SMB servers allow only FILE_SHARE_WRITE for the shared access type on a mailslot.

<40> Section 2.2.8: Windows-based SMB servers allow only the following options on a named pipe: FILE_OPEN, FILE_CREATE, and FILE_OPENIF. Windows-based SMB servers allow only the FILE_CREATE option on a mailslot.

<41> Section 2.2.8: Windows-based SMB servers force off the FILE_SYNCHRONOUS_IO_ALERT, FILE_SYNCHRONOUS_IO_NONALERT, and FILE_CREATE_TREE_CONNECTION options when sending this request. Windows-based SMB servers fail requests with the FILE_OPEN_BY_FILE_ID option set and return STATUS_NOT_SUPPORTED in the **Status** field of the SMB header in the server response. Windows-based SMB servers fail requests when both the FILE_DIRECTORY_FILE and FILE_NON_DIRECTORY_FILE options are set and return STATUS_NOT_SUPPORTED in the **Status** field of the SMB header in the server response.Windows-based SMB servers allow only a combination of the following options on a named pipe or a mailslot: FILE_WRITE_THROUGH, FILE_SYNCHRONOUS_IO_ALERT, and FILE_SYNCHRONOUS_IO_NONALERT.

<42> Section 2.2.8: Windows clients do not always set this flag to 0.

<43> Section 2.2.8: Windows clients do not always set this flag to 0.

<44> Section 2.2.8: Windows-based SMB servers on Windows NT 4.0 do not support an **ImpersonationLevel** field that is set to SECURITY_DELEGATION.

<45> Section 2.2.8: Windows-based SMB servers ignore the value passed in the client request and treat the request as if static tracking was requested and the SECURITY_EFFECTIVE_ONLY bit field was set.

<46> Section 2.2.9: If the **Name** field in the client request specifies a multilevel directory path that requires creating more than one new directory level, Windows-based SMB servers fail the request and return STATUS_OBJECT_PATH_NOT_FOUND.

<47> Section 2.2.9: Windows-based SMB servers do not grant opportunistic locks (oplocks) if the **DesiredAccess** field in the SMB_COM_NT_CREATE_ANDX client request includes DELETE access, or if the **CreateOptions** field in the SMB_COM_NT_CREATE_ANDX client request has the FILE_DIRECTORY_FILE option set. Otherwise, the type of oplock that the Windows-based SMB server attempts to grant is based both on the **Flags** field in the SMB_COM_NT_CREATE_ANDX client request and on whether the client supports CAP_LEVEL_II_OPLOCKS (the LEVEL_II_OPLOCKS bit is set in the **Capabilities** field in the SMB_COM_SESSION_SETUP_ANDX client request (section 2.2.4)).

<48> Section 2.2.9: Windows-based SMB servers typically return a value that is a multiple of the sector or cluster size of the underlying physical device.

<49> Section 2.2.9: Windows-based servers put 0 in the FileId and **VolumeGUID** fields; and these values are ignored by Windows-based SMB clients.

<50> Section 2.2.9: Windows supports the notion of a guest account and sets this field appropriately for the defined guest account rights.

<51> Section 2.2.11: Windows-based servers do not implement this feature and set the **ServerFid** field to 0.

<52> Section 2.2.11: Windows supports the notion of a guest account and sets this field based on the defined guest account rights on the server.

<53> Section 2.2.12: Windows-based SMB clients never send the TRANS_MAILSLOT_WRITE client request. Windows-based SMB clients use the Remote Mailslot Protocol to send the TRANS_MAILSLOT_WRITE subcommand to Windows-based SMB servers. For more information about the Remote Mailslot Protocol, see [MS-MAIL] and [MSLOT]. Windows-based SMB servers receive the TRANS_MAILSLOT_WRITE client request and write the data to the mailslot.

<54> Section 2.2.12.1: Windows-based SMB clients set the **Flags** field to 0 for this request.

<55> Section 2.2.12.1: Windows-based SMB clients set the time-out value to 0 for this request. Windows-based SMB servers ignore the **Timeout** field on this request.

<56> Section 2.2.12.1: When the **DataCount** field is zero, Windows-based SMB clients set the **DataOffset** field to a location in the packet following the Parameter bytes (**ParameterOffset** plus **ParameterCount**).

<57> Section 2.2.12.1: Windows-based clients set this field to the word PIPE with a backslash before and after PIPE (the \PIPE\ string).

<58> Section 2.2.12.2: When the **ParameterCount** field is 0, Windows-based SMB servers set **ParameterOffset** to a location in the packet where the Parameter bytes would have been placed if present.

<59> Section 2.2.12.2: Windows-based servers set the **DataOffset** field to the location in the packet following the parameter bytes (**ParameterOffset** plus **ParameterCount**).

<60> Section 2.2.12.3: Windows-based SMB clients do not use this method of reading raw data from a named pipe that ignores message boundaries if the pipe was set up as a message mode pipe.

<61> Section 2.2.12.3: Windows-based SMB clients set the **Flags** field to 0 for this request.

<62> Section 2.2.12.3: Windows-based SMB clients set the time-out value to 0 for this request. Windows-based SMB servers ignore the **Timeout** field on this request.

<63> Section 2.2.12.3: When the **ParameterCount** field is 0, Windows-based SMB clients set **ParameterOffset** to a location in the packet following the **Name** field.

<64> Section 2.2.12.3: When the **DataCount** field is 0, Windows-based SMB clients set the **DataOffset** field to a location in the packet following the Parameter bytes (**ParameterOffset** plus **ParameterCount**).

<65> Section 2.2.12.3: Windows-based clients set this field to the word PIPE with a backslash before and after PIPE (the \PIPE\ string).

<66> Section 2.2.12.4: When the **ParameterCount** field is 0, Windows-based SMB servers set **ParameterOffset** to a location in the packet where Parameter bytes would have been placed, if present.

<67> Section 2.2.12.4: Windows-based servers set the **DataOffset** field to the location in the packet following the parameter bytes (**ParameterOffset** plus **ParameterCount**) unless this is a transaction response that is being split into multiple response packets. If it is being split into multiple response packets and this particular response packet holds no data bytes (the **DataCount** field is 0), the **DataOffset** field will be set to 0.

<68> Section 2.2.12.5: Windows-based SMB clients set the **Flags** field to 0 for this request.

<69> Section 2.2.12.5: Windows-based SMB clients set the time-out value to 0 for this request. Windows-based SMB servers ignore the **Timeout** field on this request.

<70> Section 2.2.12.5: When the **ParameterCount** field is 0, Windows-based SMB clients set **ParameterOffset** to a location in the packet following the **Name** field.

<71> Section 2.2.12.5: When the **DataCount** field is 0, Windows-based SMB clients set the **DataOffset** field to a location in the packet following the Parameter bytes (**ParameterOffset** plus **ParameterCount**).

<72> Section 2.2.12.5: Windows-based clients set this field to the word PIPE with a backslash before and after PIPE (the \PIPE\ string).

<73> Section 2.2.12.6: Windows-based servers set the **DataOffset** field to the location in the packet following the parameter bytes (**ParameterOffset** plus **ParameterCount**).

<74> Section 2.2.12.7: Windows-based SMB clients set the **Flags** field to 0 for this request.

<75> Section 2.2.12.7: Windows-based SMB clients set the time-out value to 0 for this request. Windows-based servers ignore the **Timeout** field on this request.

<76> Section 2.2.12.7: When the **DataCount** field is 0, Windows-based SMB clients set the **DataOffset** field to a location in the packet following the Parameter bytes (**ParameterOffset** plus **ParameterCount**).

<77> Section 2.2.12.7: Windows-based clients set this field to the word PIPE with a backslash before and after PIPE (the \PIPE\ string).

<78> Section 2.2.12.8: When the **ParameterCount** field is 0, Windows-based SMB servers set **ParameterOffset** to a location in the packet where Parameter bytes would have been placed, if present.

<79> Section 2.2.12.9: Windows-based SMB clients set the **Flags** field to 0 for this request.

<80> Section 2.2.12.9: Windows-based SMB clients set the time-out value to 0 for this request. Windows-based SMB servers ignore the **Timeout** field on this request.

<81> Section 2.2.12.9: When the **DataCount** field is 0, Windows-based SMB clients set the **DataOffset** field to a location in the packet following the Parameter bytes (**ParameterOffset** plus **ParameterCount**).

<82> Section 2.2.12.9: Windows-based clients set this field to the word PIPE with a backslash before and after PIPE (the \PIPE\ string).

<83> Section 2.2.12.10: Windows-based servers set the **DataOffset** field to the location in the packet following the parameter bytes (**ParameterOffset** plus **ParameterCount**) unless this is a transaction response that is being split into multiple response packets. If it is being split into multiple response packets and this particular response packet holds no data bytes (the **DataCount** field is 0), then the **DataOffset** field will be set to 0.

<84> Section 2.2.12.11: Windows-based SMB clients set the **Flags** field to 0 for this request.

<85> Section 2.2.12.11: Windows-based SMB clients set the time-out value to 0 for this request. Windows-based SMB servers ignore the **Timeout** field on this request.

<86> Section 2.2.12.11: When the **DataCount** field is 0, Windows-based SMB clients set the **DataOffset** field to a location in the packet following the Parameter bytes (**ParameterOffset** plus **ParameterCount**).

<87> Section 2.2.12.11: Windows-based clients set this field to the word PIPE with a backslash before and after PIPE (the \PIPE\ string).

<88> Section 2.2.12.12: When the **ParameterCount** field is 0, Windows-based SMB clients set **ParameterOffset** to a location in the packet following the **Name** field.

<89> Section 2.2.12.13: Windows-based SMB clients do not use this method of writing raw data into a named pipe when the named pipe is in message mode.

<90> Section 2.2.12.13: Windows-based SMB clients set the **Flags** field to 0 for this request.

<91> Section 2.2.12.13: Windows-based SMB clients set the time-out value to 0 for this request. Windows-based SMB servers ignore the **Timeout** field on this request.

<92> Section 2.2.12.13: When the **ParameterCount** field is 0, Windows-based SMB clients set **ParameterOffset** to a location in the packet following the **Name** field.

<93> Section 2.2.12.13: Windows-based clients set this field to the word PIPE with a backslash before and after PIPE (the \PIPE\ string).

<94> Section 2.2.12.14: When the **ParameterCount** field is 0, Windows-based SMB servers set **ParameterOffset** to a location in the packet where Parameter bytes would have been placed, if present.

<95> Section 2.2.12.14: Windows-based servers set the **DataOffset** field to the location in the packet following the parameter bytes (**ParameterOffset** plus **ParameterCount**).

<96> Section 2.2.12.15: Windows-based SMB clients set the **Flags** field to 0 for this request.

<97> Section 2.2.12.15: Windows-based SMB clients set the time-out value to 0 for this request. Windows-based SMB servers ignore the **Timeout** field on this request.

<98> Section 2.2.12.15: When the **ParameterCount** field is 0, Windows-based SMB clients set **ParameterOffset** to a location in the packet following the **Name** field.

<99> Section 2.2.12.15: When the **DataCount** field is 0, Windows-based SMB clients set the **DataOffset** field to a location in the packet following the Parameter bytes (**ParameterOffset** plus **ParameterCount**).

<100> Section 2.2.12.15: Windows-based clients set this field to the word PIPE with a backslash before and after PIPE (the \PIPE\ string).

<101> Section 2.2.12.16: When the **ParameterCount** field is 0, Windows-based SMB servers set **ParameterOffset** to a location in the packet where Parameter bytes would have been placed, if present.

<102> Section 2.2.12.16: Windows-based servers set the **DataOffset** field to the location in the packet following the parameter bytes (**ParameterOffset** plus **ParameterCount**) unless this is a transaction response that is being split into multiple response packets. If it is being split into multiple response packets and this particular response packet holds no data bytes (the **DataCount** field is 0), then the **DataOffset** field will be set to 0.

<103> Section 2.2.12.17: Windows-based SMB clients set the **Flags** field to 0 for this request.

<104> Section 2.2.12.17: Windows-based SMB clients set the time-out value to 0 for this request. Windows-based SMB servers ignore the **Timeout** field on this request.

<105> Section 2.2.12.17: Windows-based clients set this field to the word PIPE with a backslash before and after PIPE (the \PIPE\ string).

<106> Section 2.2.12.18: When the **ParameterCount** field is 0, Windows-based SMB servers set **ParameterOffset** to a location in the packet where Parameter bytes would have been placed, if present.

<107> Section 2.2.12.18: Windows-based servers set the **DataOffset** field to the location in the packet following the parameter bytes (**ParameterOffset** plus **ParameterCount**.

<108> Section 2.2.12.19: Windows-based SMB clients set the **Flags** field to 0 for this request.

<109> Section 2.2.12.19: Windows-based SMB clients specify a time-out if requested to do so by the application that is requesting the TRANS_WAIT_NMPIPE operation.

<110> Section 2.2.12.19: When the **DataCount** field is 0, Windows-based SMB clients set the **DataOffset** field to a location in the packet following the Parameter bytes (**ParameterOffset** plus **ParameterCount**).

<111> Section 2.2.12.19: Windows-based SMB clients always sets it to 0 and Windows-based SMB servers ignore this field when the request is received.

<112> Section 2.2.12.20: When the **ParameterCount** field is 0, Windows-based SMB servers set ParameterOffset to a location in the packet where Parameter bytes would have been placed, if present.

<113> Section 2.2.12.20: Windows-based servers set the **DataOffset** field to the location in the packet following the parameter bytes (ParameterOffset plus **ParameterCount**.

<114> Section 2.2.12.21: Windows-based clients never send this request.

<115> Section 2.2.12.21: Windows-based SMB clients set the **Flags** field to 0 for this request.

<116> Section 2.2.12.21: Windows-based SMB clients do not send this request.

<117> Section 2.2.12.21: Windows-based clients set this field to the word PIPE with a backslash before and after PIPE (the \PIPE\ string).

<118> Section 2.2.12.22: When the ParameterCount field is 0, Windows-based SMB servers set ParameterOffset to a location in the packet where Parameter bytes would have been placed if present.

<119> Section 2.2.12.22: Windows-based servers set the **DataOffset** field to the location in the packet following the parameter bytes (**ParameterOffset** plus **ParameterCount**.

<120> Section 2.2.12.23: Windows-based SMB clients never send the TRANS_MAILSLOT_WRITE client request. Windows-based SMB clients use the Remote Mailslot Protocol to send the TRANS_MAILSLOT_WRITE subcommand to Windows-based SMB servers. For more information about the Remote Mailslot Protocol, see [MS-MAIL] and [MSLOT].

<121> Section 2.2.12.23: Windows-based SMB clients set the **Flags** field to 0 for this request.

<122> Section 2.2.12.23: Windows-based SMB servers ignore the **Timeout** field when the request is received.

<123> Section 2.2.12.23: Windows-based SMB servers ignore the **Priority** field when the request is received.

<124> Section 2.2.12.23: Windows-based SMB servers ignore the **Class** field when the request is received.

<125> Section 2.2.12.24: If the client request lacks permission to open the specific mailslot, the Windows-based SMB server returns STATUS_ACCESS_DENIED as the error code in the **Status** field of the SMB header.

<126> Section 2.2.12.24: Windows-based servers set the **DataOffset** field to the location in the packet following the parameter bytes (**ParameterOffset** plus **ParameterCount**.

<127> Section 2.2.12.24: Windows-based SMB servers set the error code in the **Status** field of the SMB header. On failure, only the SMB packet header is sent to the client in the response. So the **OperationStatus** field is never sent in the response on failure.

<128> Section 2.2.13.3: Windows-based servers ignore the **Timeout** field when the request is received.

<129> Section 2.2.13.3: When the **DataCount** field is 0, Windows-based SMB clients set the **DataOffset** field to a location in the packet following the Parameter bytes (**ParameterOffset** plus **ParameterCount**).

<130> Section 2.2.13.11: Windows-based SMB clients set the **Flags** field to 0 for this request.

<131> Section 2.2.13.11: Windows-based SMB clients set the **Flags** field to 0 for this request.

<132> Section 2.2.13.11: Windows-based SMB clients set the time-out value to 0 for this request. Windows-based SMB servers ignore the **Timeout** field on this request.

<133> Section 2.2.13.12: When the ParameterCount field is 0, Windows-based SMB servers set **ParameterOffset** to a location in the packet where Parameter bytes would have been placed if present.

<134> Section 2.2.13.12: Windows-based servers set the DataOffset field to the location in the packet following the parameter bytes (ParameterOffset plus ParameterCount).

<135> Section 2.2.13.14.3: Windows-based clients do not issue TRANS2_FIND_FIRST2 requests with the special **FileName** pattern natively. There are applications that run on Windows 9x clients that MAY execute this query.

<136> Section 2.2.13.17: When the **DataCount** field is 0, Windows-based SMB clients set the DataOffset field to a location in the packet following the Parameter bytes (ParameterOffset plus ParameterCount).

<137> Section 2.2.14.1: Windows-based servers do not implement this function. They always return SUCCESS, but do not perform the rename. Windows-based clients do not issue this request.

<138> Section 2.2.14.7: A definitive list of Windows FSCTL and IOCTL control code operations is as specified in [MS-FSCC] section 2.3. The format of the corresponding data blocks for the input buffer for each individual request is as specified in [MS-FSCC] section 2.3.

<139> Section 2.2.14.7: Windows-based clients will pass through FSCTL/IOCTL operations to the server. The Windows-based server will pass through IOCTL and FSCTL requests that the server does not understand, or the server will specifically block to the underlying file system for processing. The file system MAY reject or process these requests. The IOCTL and FSCTL values expected to support specific client features are as specified in [MS-FSCC] section 2.3.

<140> Section 2.2.14.8: For a list of control codes, see [MS-FSCC] section 2.3. For details on corresponding data formats for FSCTLs and IOCTLs implemented by Windows systems, see [MS-FSCC] section 2.3.

<141> Section 2.2.14.8.2: Currently, all Windows Server 2003 and later servers set **ContextLength** to 0.Windows-based clients ignore the **ContextLength**. All other Windows-based servers fail the request with STATUS_NOT_SUPPORTED.

<142> Section 2.2.17: For a detailed listing of possible status codes available on Windows implementations, see [MS-FSCC] section 2.2.

<143> Section 3.1.3: SupportsExtendedSecurity is TRUE for Windows 2000 and later. For all other Windows SMB clients and servers, it is FALSE.

Message signing is supported on Windows NT 4.0 SP6 and later, and for Windows 2000 and later. MessageSigningPolicy is based on system configuration. The default is that Windows SMB clients have message signing enabled, and Windows SMB servers have message signing disabled. The exception is that: The default for Windows 2000 domain controllers is to have message signing enabled for SMB servers, and the default for Windows Server 2003 domain controllers is to have the message signing required for SMB servers.

<144> Section 3.1.5.1: For client-specific and server-specific behavior on Windows, see the Windows Behavior sections in sections 3.2.5.1 and 3.3.5.1.

<145> Section 3.2.4.2.1: The Windows implementation, by default, attempts to connect on all available SMB transports (NetBIOS-compatible and direct TCP) simultaneously and selects the one that succeeds the fastest. Any connection that is not selected is immediately closed. Windows also allows an upper layer to specify what transport to use.

<146> Section 3.2.4.2.2: Windows XP SP3 and Windows Vista based SMB clients will set the SMB_FLAGS2_SMB_SECURITY_SIGNATURE_REQUIRED bit in the **Flags2** field of the SMB header of a Session Setup request.

<147> Section 3.2.4.2.3:

- Windows 2000 and later support extended security.

- Windows systems implement the first option above.

<148> Section 3.2.4.2.3.1: Windows-based SMB clients are configured by default not to send plaintext passwords. Sending plain text passwords can be configured via a registry setting.

<149> Section 3.2.4.2.4: Windows 2000 and later establish a new tree connect for each session , even if the target share is the same. Windows NT 4.0 and earlier reuse an established tree connect for multiple sessions .

<150> Section 3.2.4.2.4: Windows Server 2003 and later request signing key protection.

<151> Section 3.2.4.2.4: Windows 2000and later request extended information responses.

<152> Section 3.2.4.3: Windows NT 4.0 and later clients issue an SMB_COM_NT_CREATE_ANDX for the NT LM 0.12 dialect for which all the extensions here are described. Earlier Win9x-based clients use SMB_COM_OPEN_ANDX.

<153> Section 3.2.4.3.1: Windows XP and later scan paths and set the flag appropriately.

<154> Section 3.2.4.3.1: No Windows-based clients use this flag.

<155> Section 3.2.4.4: Windows NT 4.0 and later clients issue large reads if the server supports them. Windows NT 4.0 and later servers ignore MaxCountHigh if set to 0xFFFF for file reads. Windows NT 4.0 and later clients set the **Timeout** field to 0xFFFFFFFF on pipe reads.

<156> Section 3.2.4.5: Windows NT 4.0and later clients set this field to 0 for non message mode pipe writes.

<157> Section 3.2.4.5: Windows NT 4.0 and later clients issue large writes of up to 64 KB.

<158> Section 3.2.4.6: Windows-based clients pass through these requests to the server regardless of the information level provided in the request.

<159> Section 3.2.4.7: Windows-based clients use both TRANS2_QUERY_FILE_INFORMATION and TRANS2_QUERY_PATH_INFORMATION, depending on if they want to execute on a previously opened handle or on using a file path. Clients based on Windows 2000and later use pass-through queries, if available.

<160> Section 3.2.4.8: Windows-based clients use both TRANS2_SET_FILE_INFORMATION and TRANS2_SET_PATH_INFORMATION, depending on if they want to execute on a previously opened handle or on using a file path. Clients based on Windows 2000 and later use pass-through queries, if available.

<161> Section 3.2.4.9: Clients based on Windows 2000 and later use pass-through queries, if available.

<162> Section 3.2.4.10: Windows 2000 and later clients use pass-through queries, if available.

<163> Section 3.2.4.11.1: Windows XP-based clients use this FSCTL. Windows NT 4.0 and Windows 2000-based clients MAY use this FSCTLif the previous versions down-level application is installed on them. Windows 9x-based clients do not use this method if the previous versions down-level application is installed on them; instead, they use the FindFirst method for enumeration, as specified in section 3.2.4.6.

<164> Section 3.2.4.12: Windows NT 4.0 SP6a, Windows 9x-based clients, Windows XP SP2 with client update (as specified in KB article number 898900) and Windows Server 2003 SP1 and later clients support DFS.

<165> Section 3.2.4.13: SMB clients and servers run on Windows 2000 and later support the NT_TRANS_QUERY_QUOTA subcommand.

<166> Section 3.2.5.1: Windows-based clients that use message signing disconnect the connection on receipt of an incorrectly signed message.

<167> Section 3.3.2: For the Windows implementation of Kerberos, the default expiration time is eight hours.

<168> Section 3.3.5.2: SMB servers run on Windows 2000and later support Extended Security, and they all are configured to use SPNEGO (as specified in [RFC4178]) as their GSS authentication protocol. Windows operating systems that use extended security send a GSS token (or fragment) if their SPNEGO implementation supports it. For details on Windows behavior, see [RFC4178].

<169> Section 3.3.5.5: SMB servers on Windows 2000 and later use extended responses if the client requests them, and return 0 for the VolumeGUID and FileId.

<170> Section 3.3.5.5: SMB servers on Windows Server 2003and later support the REPARSE flag and previous version access. SMB server on Windows Server 2003 and later parse paths when the flag is not set only when configured to do so, and not by default. This is used to expose previous version logic to applications running on clients whose SMB client does not understand the REPARSE flag and does not set it appropriately.

<171> Section 3.3.5.6: SMB servers on Windows 2000 and later use extended responses if the client requests them.

<172> Section 3.3.5.7: SMB servers on Windows NT 4.0and later allow for large reads of 64 KB, but no larger. Requests for larger values return, at most, 64 KB.

<173> Section 3.3.5.9: Windows NTand later support these new information levels for directory queries.

<174> Section 3.3.5.9: Windows Server 2003 supports previous versions but does not support this method of enumerating them, by default. The capability can be configured to be active by the administrator. The purpose is to allow an application (on a client that does not support the IOCTL command) to have a method of enumerating previous versions.

<175> Section 3.3.5.17.3: Windows Behavior: This operation is valid only on Windows Server 2003 SP1and later.

Windows Server 2003 SP1 server default values and limits are as follows:

- Maximum number of chunks the server accepts to be copied in a single request is 256.
- Maximum amount of data the server accepts to be copied in a single chunk is 1 MB.
- Maximum amount of data the server accepts to be copied in a single request is 16 MB.
- Time-out for server-side data copy operations to complete is 25 seconds.

7 Index

P

R

S

T

V

W