[MS-SMB2]:
Server Message Block (SMB)
Version 2.0 Protocol Specification

Revision Summary

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

Date	Revision History	Revision Class	Comments
05/11/2007	1.3		Monthly release
06/01/2007	1.3.1	Editorial	Revised and edited the technical content.
07/03/2007	2.0	Major	MLonghorn+90
07/20/2007	3.0	Major	Updated and revised the technical content.
08/10/2007	4.0	Major	Updated and revised the technical content.
09/28/2007	5.0	Major	Updated and revised the technical content.
10/23/2007	6.0	Major	Updated and revised the technical content.
11/30/2007	7.0	Major	Updated and revised the technical content.
01/25/2008	7.0.1	Editorial	Revised and edited the technical content.

Table of Contents

1 Introduction

This specification describes the Server Message Block (SMB) Version 2.0 Protocol, which supports the sharing of file and print resources between machines. The protocol borrows and extends concepts from the Server Message Block (SMB) Protocol, as specified in [MS-SMB]. This specification assumes familiarity with the SMB Protocol, as specified in [MS-SMB], and with the Windows security concepts that are specified in [MS-SECO].

1.1 Glossary

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

@GMT Token
Discretionary Access Control List (DACL)
Distributed File System (DFS)
File System Control (FSCTL)
Globally Unique Identifier (GUID)
Main Stream
Named Stream
Security Descriptor
Security Identifier (SID)
Snapshot
System Access Control List (SACL)
Unicode

The following terms are specific to this document:

Authenticated Context: The runtime state that is associated with the successful authentication of a security principal between the client and the server, such as the security principal itself, the cryptographic key that was generated during authentication, and the rights and privileges of this security principal.

Compounded Requests and Responses: A method of combining multiple SMB 2.0 Protocol requests or responses into a single transmission request for submission to the underlying transport.

Connection: Either a TCP or NetBIOS over TCP connection between an SMB 2.0 Protocol client and an SMB 2.0 Protocol server.

Create Context: A variable-length attribute that is sent with an SMB2 CREATE Request (section 2.2.13) or SMB2 CREATE Response that either gives extra information about how the create shall be processed, or returns extra information about how the create was processed. See sections 2.2.13.2 and 2.2.14.2.

Credit: A value that is granted to an SMB 2.0 Protocol client by an SMB 2.0 Protocol server that limits the number of outstanding requests that a client can send to a server.

Durable Open: An **open** to a file or named pipe that allows the client to attempt to preserve and reestablish the **open** after a network disconnect.

I/O Control (IOCTL): A command that is issued to a target file system or target device in order to query or alter the behavior of the target; or to query or alter the data and attributes that are associated with the target or the objects that are exposed by the target.

Open: A runtime object that corresponds to a currently established access to a specific file or named pipe from a specific client to a specific server, using a specific user security context. Both clients and servers maintain **opens** that represent active accesses.

Oplock: An opportunistic lock, or **oplock**, is a mechanism that is designed to allow clients to dynamically alter their buffering strategy in a consistent manner in order to increase performance and reduce network use. The network performance for remote file operations may be increased if a client can locally buffer file data, which reduces or eliminates the need to send and receive network packets. For example, a client may not have to write information into a file on a remote server if the client knows that no other process is accessing the data. Likewise, the client may buffer read-ahead data from the remote file if the client knows that no other process is writing data to the remote file.

There are three types of **oplocks**:

- An exclusive **oplock** allows a client to **open** a file for exclusive access and allows the client to perform arbitrary buffering.
- A batch **oplock** allows a client to keep a file **open** on the server even though the local accessor on the client machine has closed the file.
- A Level II **oplock** indicates that there are multiple readers of a file and no writers.

When a client **opens** a file, it requests that the server grant it a particular type of **oplock** on the file. The response from the server indicates the type of **oplock** that is granted to the client. The client uses the granted **oplock** type to adjust its buffering policy.

Oplock Break: An unsolicited request that is sent by an SMB 2.0 Protocol server to an SMB 2.0 Protocol client to inform the client to change the **oplock** level for a file.

Sequence Number: A number that uniquely identifies a request and response that is sent on an SMB 2.0 Protocol **connection**. For a description of how **sequence numbers** are allocated, see sections 3.2.1.1 and 3.3.1.2.

Session: An **authenticated context** that is established between an SMB 2.0 Protocol client and an SMB 2.0 Protocol server over an SMB 2.0 Protocol **connection** for a specific security principal. There could be multiple active **sessions** over a single SMB 2.0 Protocol **connection**. The **SessionId** field in the SMB2 packet header (section 2.2.1) distinguishes the various **sessions**.

Share: A local resource that is offered by an SMB 2.0 Protocol server for access by SMB 2.0 Protocol clients over the network. The SMB 2.0 Protocol defines three types of **shares**: file (or disk) **shares**, which represent a directory tree and its included files; pipe **shares**, which expose access to named pipes; and print **shares**, which provide access to print resources on the server. A pipe **share** as defined by the SMB 2.0 Protocol must always have the name "IPC$". A pipe **share** must only allow named pipe operations and DFS referral requests to itself.

Tree Connect: A connection by a specific **session** on an SMB 2.0 Protocol client to a specific **share** on an SMB 2.0 Protocol server over an SMB 2.0 Protocol **connection**. There could be multiple **tree connects** over a single SMB 2.0 Protocol **connection**. The **TreeId** field in the SMB2 packet header (section 2.2.1) distinguishes the various **tree connects**.

WorldSid: A **SID** with the specific value of S-1-1-0.

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.

[FIPS180-2] Federal Information Processing Standards Publication, "Secure Hash Standard", FIPS PUB 180-2, August 2002, http://csrc.nist.gov/publications/fips/fips180-2/fips180-2.pdf

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

[MS-SECO] Microsoft Corporation, "Windows Security Overview", January 2007.

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

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

[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

[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

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

1.2.2 Informative References

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

[MSDN-IMPERS] Microsoft Corporation, "Impersonation" http://msdn2.microsoft.com/en-us/library/ms691341.aspx

[OFFLINE] Microsoft Corporation, "Offline Files", January 2005, http://technet2.microsoft.com/WindowsServer/en/Library/830323a2-23ca-4875-af3c-06671d68ca9a1033.mspx

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

[STREAMS] Microsoft Corporation, "File Streams", http://msdn2.microsoft.com/en-us/library/aa364404.aspx

[TIMEWARP] Microsoft Corporation, "Shadow Copies for Shared Folders Tools and Settings", March 2003, http://technet2.microsoft.com/WindowsServer/en/library/78203fa0-7d7d-45d2-a8be-89fa5fd18db31033.mspx

1.3 Protocol Overview (Synopsis)

The Server Message Block (SMB) Version 2.0 Protocol is an extension of the original Server Message Block (SMB) Protocol (as specified in [MS-SMB] and [CIFS]). Both protocols are used by clients to request file and print services from a server system over the network. Both are stateful protocols in which clients establish a **connection** to a server, establish an **authenticated context** on that connection, and then issue a variety of requests to access files, printers, and named pipes for interprocess communication.

The SMB 2.0 Protocol is a major revision of the existing SMB Protocol, as specified in [MS-SMB]. The packet formats are completely different from those of the SMB Protocol; however, many of the underlying concepts are carried over. The underlying transports that are used to initiate and accept connections are either Direct TCP or NetBIOS over TCP transports as specified in [MS-SMB] section 2.1.

To retain compatibility with existing clients and servers, the existing SMB Protocol can be used to negotiate the use of the SMB 2.0 Protocol, as described in section 1.7. However, the two protocols will never be intermixed on a specified connection after one is selected during negotiation.

Like its predecessor, which was the original SMB Protocol (as specified in [MS-SMB]), the SMB 2.0 Protocol supports the following features:

- Establishing one or more authenticated contexts for different security principals on a connection.
- Connecting to multiple **shared** resources on the target server on a connection.

- Opening, reading, modifying, or closing multiple files or named pipes on the target server.
- Using the opportunistic locking of files to allow clients to cache data for better performance.
- Querying and applying attributes to files or volumes on the target server.
- Canceling outstanding operations.
- Passing through IO control code operations to the file system on the server machine.

The SMB 2.0 Protocol provides several enhancements in addition to the above features:

- Allowing an **open** to a file to be reestablished after a client connection becomes temporarily disconnected.
- Allowing the server to balance the number of simultaneous operations that a client can have outstanding at any time.
- Providing scalability in terms of the number of shares, users, and simultaneously open files.

1.4 Relationship to Other Protocols

The Server Message Block (SMB) Version 2.0 Protocol may be negotiated by using an SMB negotiate, as specified in [MS-SMB] section 1.7. After the SMB 2.0 Protocol is selected during negotiation, all messages that are sent on the connection, including the negotiate response, will be SMB 2.0 Protocol messages, as specified in this document, and no further SMB traffic will be exchanged on the connection.

For authentication, the SMB 2.0 Protocol relies on Simple and Protected GSS-API Negotiation (SPNEGO), as specified in [RFC4178] and [MS-SPNG], which in turn may rely on the Kerberos Protocol Extensions (as specified in [MS-KILE]) or the NT LAN Manager (NTLM) Authentication Protocol (as specified in [MS-NLMP]).

The SMB 2.0 Protocol uses either TCP or NetBIOS over TCP as underlying transports.

The SMB 2.0 Protocol relies on the Distributed File System (DFS): Referral Protocol as specified in [MS-DFSC] to resolve names from a namespace distributed across many servers and geographies into local names on specific file servers.

The Remote Procedure Call Protocol Extensions, as specified in [MS-RPCE], define an RPC over SMB Protocol or SMB 2.0 Protocol sequence that can use SMB 2.0 Protocol named pipes as its underlying transport. The selection of protocol is based on client behavior during negotiation, as specified in section 1.7.

1.5 Prerequisites/Preconditions

The Server Message Block (SMB) Version 2.0 Protocol assumes the availability of the following resources:

- Either TCP or NetBIOS over TCP to support reliable, in-order message delivery.
- An underlying local resource, such as a file system on the server side, exposing file, named-pipe, or printer objects.
- Infrastructure that supports Simple and Protected GSS-API Negotiation (SPNEGO), as specified in [RFC4178] and [MS-SPNG], on both the client and the server.

1.6 Applicability Statement

The Server Message Block (SMB) Version 2.0 Protocol<1> is applicable for all scenarios that involve transferring files between client and server. The SMB 2.0 Protocol is also applicable for inter-process communication between client and server using named pipes.

The SMB 2.0 Protocol may be more applicable than the SMB Protocol in scenarios that require the following features:

- Higher scalability of the number of files that a client may open simultaneously, as well as the number of shares and user **sessions** that servers may maintain.
- Quality of Service guarantees from the server for the number of requests that can be outstanding against a server at any specified time.
- Symbolic link support.
- Stronger end-to-end data integrity protection, using the SHA256 hash algorithm.
- Improved throughput across networks that have disparate characteristics.
- Improved resilience to intermittent losses of network connectivity.

1.7 Versioning and Capability Negotiation

This document covers versioning in the following areas:

- Supported Transports: This protocol can be implemented on top of NetBIOS or TCP, as defined in section 2.1.
- Protocol Versions: This protocol supports several capability bits. These are defined in section 2.2.5.
- Security and Authentication Methods: The SMB 2.0 Protocol supports authentication through the use of the Generic Security Service Application Programming Interface (GSS-API), as specified in [MS-SPNG].
- Capability Negotiation: Though the semantics and the command set for the SMB 2.0 Protocol closely match the SMB Protocol, as specified in [MS-SMB], the wire format for SMB 2.0 Protocol packets is different from that of the SMB Protocol. For maintaining interoperability between clients and servers in a mixed SMB 2.0/SMB Protocol environment, the SMB 2.0 Protocol can be negotiated in one of two ways:
  - By using an SMB negotiate message (as specified in [MS-SMB] sections 2.2.2 and 3.2.4.2.2) via a new dialect string "SMB 2.002".
  - By using an SMB2 negotiate request, as specified in section 2.2.3.

If a client uses an SMB negotiate message to indicate to an SMB 2.0 Protocol–capable server that it wants to use SMB 2.0, the server must respond with an SMB2 negotiate response as specified in section 2.2.4.

A client that maintains a runtime cache for each server with which it communicates, including whether the server is SMB 2.0 Protocol–capable, would then use an SMB2 negotiate request (as specified in section 2.2.3) in future attempts to connect to any server whose cached entry indicates support for the SMB 2.0 Protocol.

Servers capable of only the SMB 2.0 Protocol would reject communication with traditional SMB Protocol clients that do not offer "SMB 2.002" as a negotiate dialect, and accept communication only from SMB 2.0 Protocol clients.

There is currently a single version of the SMB 2.0 Protocol. Negotiating this protocol implies support for the requests and responses as specified in this document, unless specifically noted otherwise. Certain functionalities are negotiated through the exchange of capabilities as specified in sections 2.2.4 and 2.2.5.

1.8 Vendor-Extensible Fields

There are no vendor-extensible fields for the Server Message Block (SMB) Version 2.0 Protocol.

1.9 Standards Assignments

This protocol shares the standards assignments of TCP port and NetBIOS-over-TCP port, as specified in [MS-SMB] section 1.9 and [RFC1002], respectively.

2 Messages

The following sections specify how Server Message Block (SMB) Version 2.0 Protocol messages are encapsulated on the wire and common SMB 2.0 Protocol data types.

2.1 Transport

The Server Message Block (SMB) Version 2.0 Protocol shares only direct TCP and NetBIOS over TCP transports and endpoints specified in the original Server Message Block (SMB) Protocol (as specified in [MS-SMB] section 2.1).

When the SMB 2.0 Protocol is negotiated on the connection, there is no inheritance of the base SMB Protocol state. The SMB 2.0 Protocol takes over the transport connection that is initially used for negotiation, and thereafter, all protocol flow on that connection MUST be SMB 2.0 Protocol.

2.2 Message Syntax

The SMB 2.0 Protocol is composed of, and driven by, message exchanges between the client and the server in the following categories:

- Protocol negotiation (SMB2 NEGOTIATE).
- User authentication (SMB2 SESSION_SETUP, SMB2 LOGOFF).
- Share access (SMB2 TREE_CONNECT, SMB2 TREE_DISCONNECT).
- File access (SMB2 CREATE, SMB2 CLOSE, SMB2 READ, SMB2 WRITE, SMB2 LOCK, SMB2 IOCTL, SMB2 QUERY_INFO, SMB2 SET_INFO, SMB2 FLUSH, SMB2 CANCEL).
- Directory access (SMB2 QUERY_DIRECTORY, SMB2 CHANGE_NOTIFY).
- Volume access (SMB2 QUERY_INFO, SMB2 SET_INFO).
- Cache coherency (SMB2 OPLOCK_BREAK).
- Simple messaging (SMB2 ECHO).

This document specifies the messages in the preceding list.

An SMB 2.0 Protocol message is the payload packet encapsulated in a transport packet.

All SMB 2.0 Protocol messages begin with a fixed-length SMB 2.0 Protocol header that is described in section 2.2.1. The SMB 2.0 Protocol header contains a command field indicating the operation code that is requested by the client or responded to by the server. An SMB 2.0 Protocol message is of variable length, depending on the Command field in the SMB 2.0 Protocol header and on whether the SMB 2.0 Protocol message is a client request or a server response.

Unless otherwise specified, multiple-byte fields (16-bit, 32-bit, and 64-bit fields) in an SMB 2.0 Protocol message MUST be transmitted in little-endian order (least-significant byte first).

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

Unless otherwise specified, all textual strings MUST be in **Unicode** version 5.0 format, as specified in [UNICODE], using the 16-bit UTF-16 form of the encoding. Textual strings with separate fields identifying the length of the string MUST NOT be null-terminated unless otherwise specified.

Unless otherwise noted, fields marked as "unused" MUST be set to 0 when being sent and MUST be ignored when received. These fields are reserved for future protocol expansion and MUST NOT be used for implementation-specific functionality.

When it is necessary to insert unused padding bytes into a buffer for data alignment purposes, such bytes MUST be set to 0 when being sent and MUST be ignored when received.

When an error occurs, a server MUST send back an SMB 2.0 Protocol error response as specified in section 2.2.2, unless otherwise noted in section 3.3.

All constants in section 2 and 3 that begin with STATUS_ have their values defined in [MS-ERREF] section 2.3.

Operations executed on a printer share are handled on the server by creating a file, and printing the contents of the file when it is closed. Unless otherwise specified, descriptions in this document concerning protocol behavior for files also apply to printers. For information about processing specific to printers, see section 2.2.13.

2.2.1 SMB2 Packet Header

The SMB2 Packet Header (also called the SMB2 header) is the header of all SMB 2.0 Protocol packets.

There are two variants of this header:

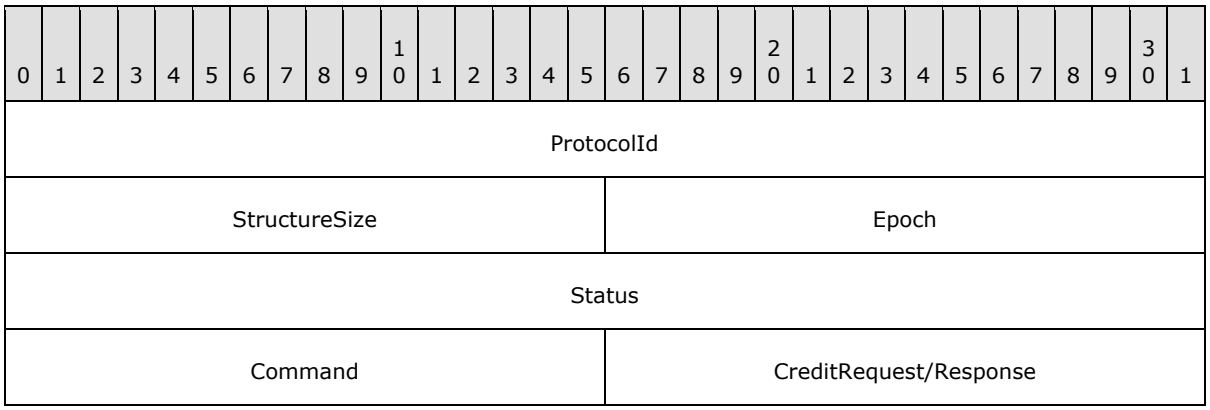
- ASYNC
- SYNC

If the SMB2_FLAGS_ASYNC_COMMAND bit is set in Flags, the header takes the form: SMB2 Protocol Header - ASYNC (section 2.2.1.1). This header format is used for responses to requests processed asynchronously by SMB2 server. For more details refer to sections 3.3.4.2, 3.2.5.1.4 and 3.2.4.23. The SMB2 CANCEL Request uses this format for cancelling requests that are being processed asynchronously.

If the SMB2_FLAGS_ASYNC_COMMAND bit is not set in Flags, the header takes the form: SMB2 Protocol Header - SYNC (section 2.2.1.2). This format is used for all requests with the exception of the SMB2 CANCEL Request to cancel a previously sent request being processed asynchronously.

2.2.1.1 SMB2 Packet Header - ASYNC

If the SMB2_FLAGS_ASYNC_COMMAND bit is set in Flags, the header takes the following form:



Flags
NextCommand
MessageId
...
AsyncId
...
SessionId
...
Signature
...
...
...

ProtocolId (4 bytes): The protocol identifier. The value must be (in network order) 0xFE, 'S', 'M', and 'B'.

StructureSize (2 bytes): MUST be set to 64, which is the size, in bytes, of the SMB2 header structure.

Epoch (2 bytes): Unused at the present, and MUST be treated as reserved. The sender MUST set this to 0, and the receiver MUST ignore it.

Status (4 bytes): The status code for a response. For a request, the client MUST set this field to 0 and the server MUST ignore it on receipt. For a response, this field can be set to any value. For a list of valid status codes, see [MS-ERREF] section 2.3.

Command (2 bytes): The command code of this packet. This field MUST contain one of the following valid commands:

Name	Value
SMB2 NEGOTIATE	0x0000
SMB2 SESSION_SETUP	0x0001
SMB2 LOGOFF	0x0002

Name	Value
SMB2 TREE_CONNECT	0x0003
SMB2 TREE_DISCONNECT	0x0004
SMB2 CREATE	0x0005
SMB2 CLOSE	0x0006
SMB2 FLUSH	0x0007
SMB2 READ	0x0008
SMB2 WRITE	0x0009
SMB2 LOCK	0x000A
SMB2 IOCTL	0x000B
SMB2 CANCEL	0x000C
SMB2 ECHO	0x000D
SMB2 QUERY_DIRECTORY	0x000E
SMB2 CHANGE_NOTIFY	0x000F
SMB2 QUERY_INFO	0x0010
SMB2 SET_INFO	0x0011
SMB2 OPLOCK_BREAK	0x0012

CreditRequest/Response (2 bytes): On a request, this field indicates the number of **credits** the client is requesting. On a response, it indicates the number of credits granted to the client. If a client does not want more credits, it MUST set this field to 1.

Flags (4 bytes): A flags field, which indicates how the operation must be processed. This field MUST be constructed using the following values:

Value	Meaning
SMB2_FLAGS_SERVER_TO_REDIR 0x00000001	When set, indicates the message is a response rather than a request. This MUST be set on responses sent from the server to the client, and MUST NOT be set on requests sent from the client to the server.
SMB2_FLAGS_ASYNC_COMMAND 0x00000002	When set, indicates that this is an asynchronously processed command.
SMB2_FLAGS_RELATED_OPERATIONS 0x00000004	When set, indicates that this command is a related operation in a compounded request chain. The use of this flag is as specified in 3.2.4.1.4.
SMB2_FLAGS_SIGNED 0x00000008	When set, indicates that this packet has been signed. The use of this flag is as specified in 3.1.5.1.
SMB2_FLAGS_DFS_OPERATIONS	When set, indicates that this command is a Distributed File

Value	Meaning
0x10000000	System (DFS) operation. The use of this flag is as specified in 3.3.5.9.

NextCommand (4 bytes): For a compounded request, this field MUST be set to the offset, in bytes, from the beginning of this SMB 2.0 Protocol header to the start of the subsequent 8-byte aligned SMB 2.0 Protocol header. If this is not a compounded request, or this is the last header in a compounded request, this value MUST be 0.

MessageId (8 bytes): A value that identifies a message request and response uniquely across all messages that are sent on the same SMB 2.0 Protocol transport connection.

AsyncId (8 bytes): A unique identification number that is created by the server to handle operations asynchronously, as specified in section 3.3.4.2.

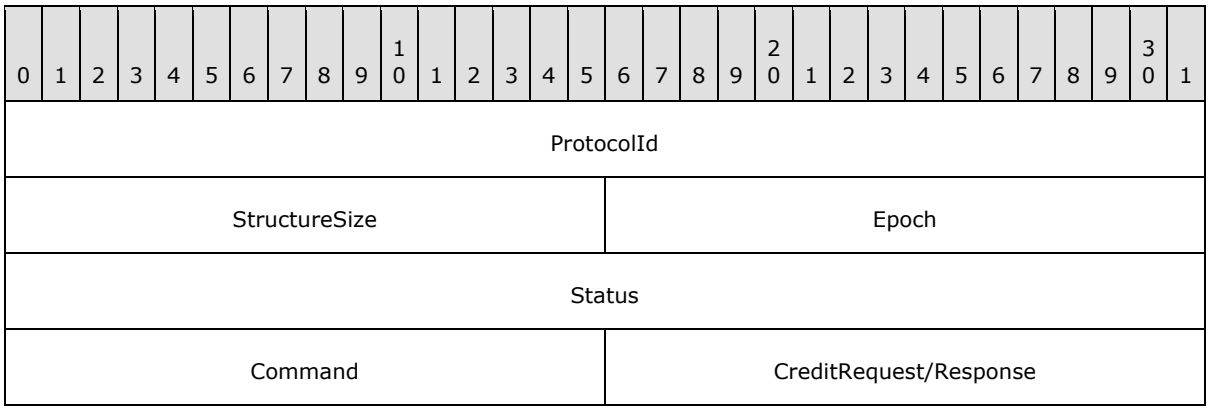
SessionId (8 bytes): Uniquely identifies the established session for the command. This MUST be 0 for requests that do not have a user context that is associated with them. This MUST be 0 for the first SMB2 SESSION SETUP request for a specified security principal. The following SMB 2.0 Protocol commands do not require the SessionId to be set to a nonzero value received from a previous SMB2 SESSION_SETUP response. *SessionId* SHOULD be set to 0 for the following commands:

- SMB2 NEGOTIATE request
- SMB2 NEGOTIATE response
- SMB2 SESSION_SETUP request
- SMB2 ECHO request
- SMB2 ECHO response

Signature (16 bytes): The 16-byte signature of the message, if SMB2_FLAGS_SIGNED is set in the **Flags** field of the SMB 2.0 Protocol header. If the message is not signed, this field MUST be 0.

2.2.1.2 SMB2 Packet Header - SYNC

If the SMB2_FLAGS_ASYNC_COMMAND bit is not set in Flags, the header takes the following form:



Flags
NextCommand
MessageId
...
ProcessId
TreeId
SessionId
...
Signature
...
...
...

ProtocolId (4 bytes): The protocol identifier. The value must be (in network order) 0xFE, 'S', 'M', and 'B'.

StructureSize (2 bytes): This MUST be set to 64, which is the size, in bytes, of the SMB2 header structure.

Epoch (2 bytes): Unused at the present, and MUST be treated as reserved. The sender MUST set this to 0, and the receiver MUST ignore it.

Status (4 bytes): The status code for a response. For a request, the client MUST set this field to 0 and the server MUST ignore it on receipt. For a response, this field can be set to any value. For a list of valid status codes, see [MS-ERREF] section 2.3.

Command (2 bytes): The command code of this packet. This field MUST contain one of the following valid commands:

Name	Value
SMB2 NEGOTIATE	0x0000
SMB2 SESSION_SETUP	0x0001
SMB2 LOGOFF	0x0002

Name	Value
SMB2 TREE_CONNECT	0x0003
SMB2 TREE_DISCONNECT	0x0004
SMB2 CREATE	0x0005
SMB2 CLOSE	0x0006
SMB2 FLUSH	0x0007
SMB2 READ	0x0008
SMB2 WRITE	0x0009
SMB2 LOCK	0x000A
SMB2 IOCTL	0x000B
SMB2 CANCEL	0x000C
SMB2 ECHO	0x000D
SMB2 QUERY_DIRECTORY	0x000E
SMB2 CHANGE_NOTIFY	0x000F
SMB2 QUERY_INFO	0x0010
SMB2 SET_INFO	0x0011
SMB2 OPLOCK_BREAK	0x0012

CreditRequest/Response (2 bytes): On a request, this field indicates the number of credits the client is requesting. On a response, it indicates the number of credits that are granted to the client.

Flags (4 bytes): A flags field, which indicates how the operation must be processed. This field MUST be constructed using the following values:

Value	Meaning
SMB2_FLAGS_SERVER_TO_REDIR 0x00000001	When set, indicates the message is a response, rather than a request. This MUST be set on responses sent from the server to the client and MUST NOT be set on requests sent from the client to the server.
SMB2_FLAGS_ASYNC_COMMAND 0x00000002	When set, indicates that this is an asynchronously processed command.
SMB2_FLAGS_RELATED_OPERATIONS 0x00000004	When set, indicates that this command is a related operation in a compounded request chain. The use of this flag is as specified in 3.2.4.1.4.
SMB2_FLAGS_SIGNED 0x00000008	When set, indicates that this packet has been signed. The use of this flag is as specified in 3.1.5.1.
SMB2_FLAGS_DFS_OPERATIONS	When set, indicates that this command is a Distributed File

Value	Meaning
0x10000000	System (DFS) operation. The use of this flag is as specified in 3.3.5.9.

NextCommand (4 bytes): For a **compounded request**, this field MUST be set to the offset, in bytes, from the beginning of this SMB 2.0 Protocol header to the start of the subsequent 8-byte aligned SMB 2.0 Protocol header. If this is not a compounded request, or this is the last header in a compounded request, this value MUST be 0.

MessageId (8 bytes): A value that identifies a message request and response uniquely across all messages that are sent on the same SMB 2.0 Protocol transport connection.

ProcessId (4 bytes): The client-side identification of the process that is issuing the request.

TreeId (4 bytes): Uniquely identifies the **tree connect** for the command. This MUST be 0 for the SMB2 TREE_CONNECT request. The **TreeId** MAY be any unsigned 32-bit integer that is received from a previous SMB2 TREE_CONNECT response. The following SMB 2.0 Protocol commands do not require the **TreeId** to be set to a nonzero value received from a previous SMB2 TREE_CONNECT response. **TreeId** SHOULD be set to 0 for the following commands:

- SMB2 NEGOTIATE request
- SMB2 NEGOTIATE response
- SMB2 SESSION_SETUP request
- SMB2 SESSION_SETUP response
- SMB2 LOGOFF request
- SMB2 LOGOFF response
- SMB2 ECHO request
- SMB2 ECHO response

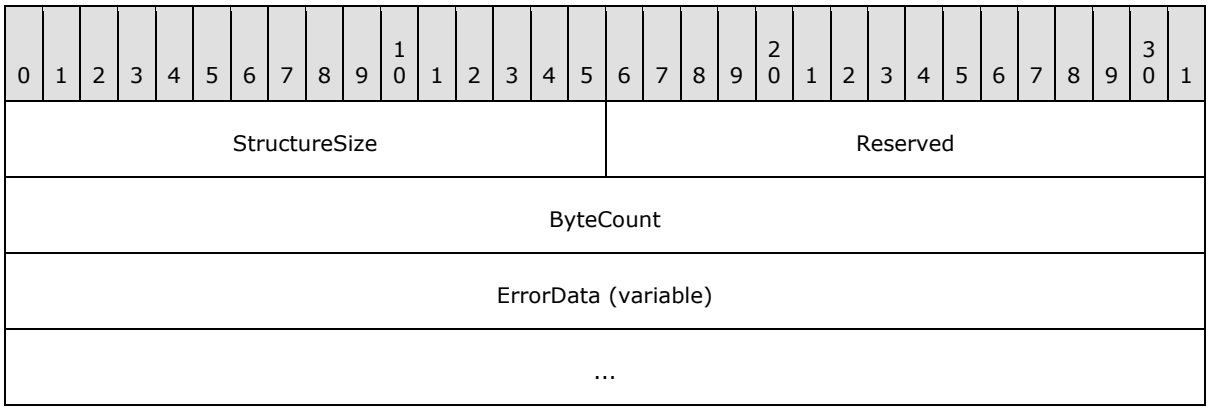
SessionId (8 bytes): Uniquely identifies the established session for the command. This MUST be 0 for requests that do not have a user context that is associated with them. This MUST be 0 for the first SMB2 SESSION_SETUP request for a specified security principal. The following SMB 2.0 Protocol commands do not require the SessionId to be set to a nonzero value received from a previous SMB2 SESSION_SETUP response. SessionId SHOULD be set to 0 for the following commands:

- SMB2 NEGOTIATE request
- SMB2 NEGOTIATE response
- SMB2 SESSION_SETUP request
- SMB2 ECHO request
- SMB2 ECHO response

Signature (16 bytes): The 16-byte signature of the message, if SMB2_FLAGS_SIGNED is set in the Flags field of the SMB 2.0 Protocol header. If the message is not signed, this field MUST be 0.

2.2.2 SMB2 ERROR Response Packet

The SMB2 ERROR Response packet is sent by the server to respond to a request that has failed or encountered an error. This response is composed of an SMB2 Packet Header (section 2.2.1) followed by this response structure:



StructureSize (2 bytes): The server MUST set this field to 9, indicating the size of the response structure, not including the header. The server MUST set it to this value regardless of how long **ErrorData**[] actually is in the response being sent.

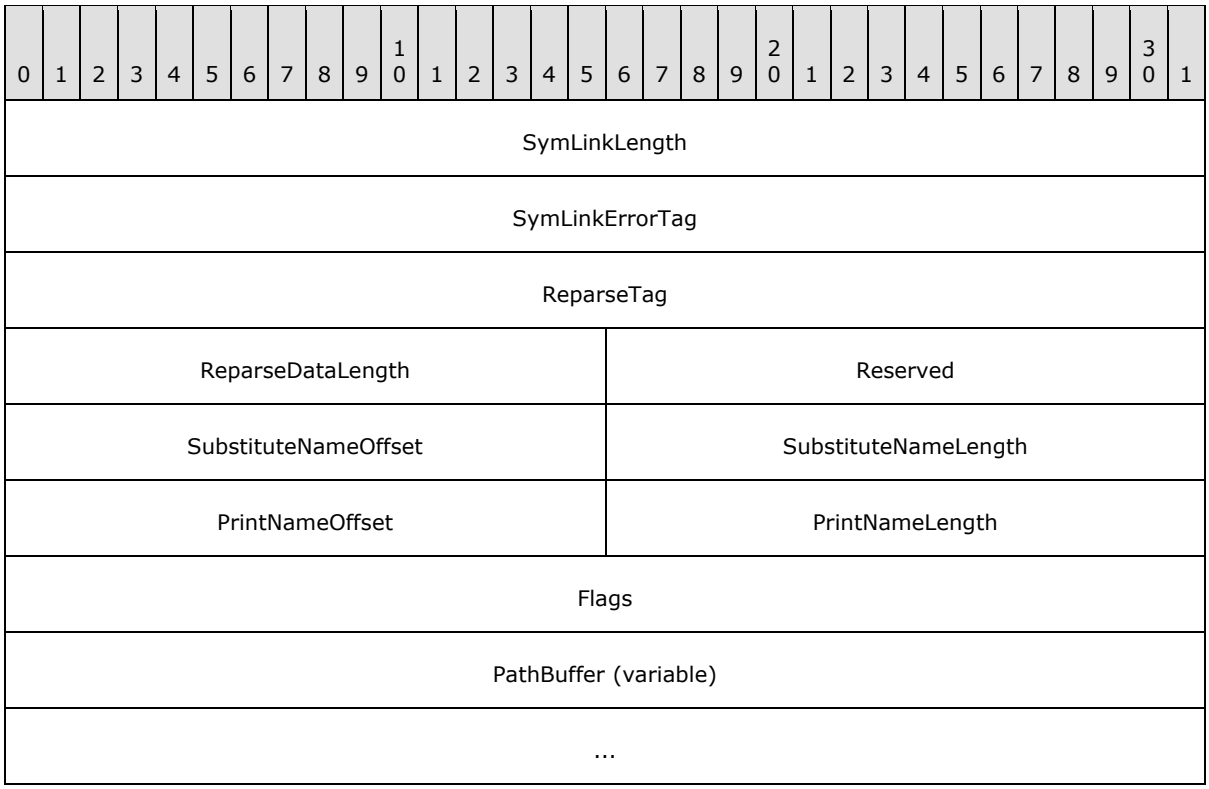
Reserved (2 bytes): Unused at the present and MUST be treated as reserved. The server MUST set this to 0, and the client MUST ignore it on receipt.

ByteCount (4 bytes): The number of bytes of data contained in **ErrorData**[].

ErrorData (variable): A variable-length data field that contains extended error information. If the Status code in the header of the response is set to STATUS_STOPPED_ON_SYMLINK, this field MUST contain a Symbolic Link Error Response as specified in section 2.2.2.1. If the **ByteCount** field is zero then the server MUST supply an **ErrorData** field that is one byte in length, and SHOULD set that byte to zero; the client MUST ignore it on receipt.<2>

2.2.2.1 Symbolic Link Error Response

The Symbolic Link Error Response is used to indicate that a symbolic link was encountered on create; it describes the target path that the client MUST use if it wants to follow the symbolic link. This structure is contained in the ErrorData section of the SMB2 ERROR Response (section 2.2.2). This structure MUST NOT be returned in an SMB2 ERROR Response unless the Status code in the header of that response is set to STATUS_STOPPED_ON_SYMLINK. The structure has the following format:



SymLinkLength (4 bytes): The length, in bytes of the response including the variable-length portion and excluding SymLinkLength.

SymLinkErrorTag (4 bytes): The server MUST set this field to 0x4C4D5953.

ReparseTag (4 bytes): The type of link encountered. The server MUST set this field to 0xA000000C.

ReparseDataLength (2 bytes): The length, in bytes, of the variable-length portion of the symbolic link error response plus the size of the static portion, not including SymLinkLength, SymLinkErrorTag, ReparseTag, ReparseDataLength, and Reserved. The server MUST set this to the size of **PathBuffer**[], in bytes, plus 12. (12 is the size of **SubstituteNameOffset**, **SubstituteNameLength**, **PrintNameOffset**, **PrintNameLength**, and **Flags**.)

Reserved (2 bytes): Unused at the present, and MUST be treated as reserved. The server MUST set this field to 0, and the client MUST ignore it.

SubstituteNameOffset (2 bytes): The offset, in bytes, from the beginning of the error response of the substitute name string in the **PathBuffer** array. The substitute name is the name the client MUST use to access this file if it wants to follow the symbolic link.

SubstituteNameLength (2 bytes): The length, in bytes, of the substitute name string. If there is a NULL termination at the end of the string, it is not included in the **SubstituteNameLength** count. This value MUST be greater than or equal to 0.

PrintNameOffset (2 bytes): The offset, in bytes, from the beginning of the error response of the print name string in the **PathBuffer** array. The print name is the user friendly name the client MUST return to the application if it requests the name of the symbolic link target.

PrintNameLength (2 bytes): The length, in bytes, of the print name string. If there is a NULL termination at the end of the string, it is not included in the **PrintNameLength** count. This value MUST be greater than or equal to 0.

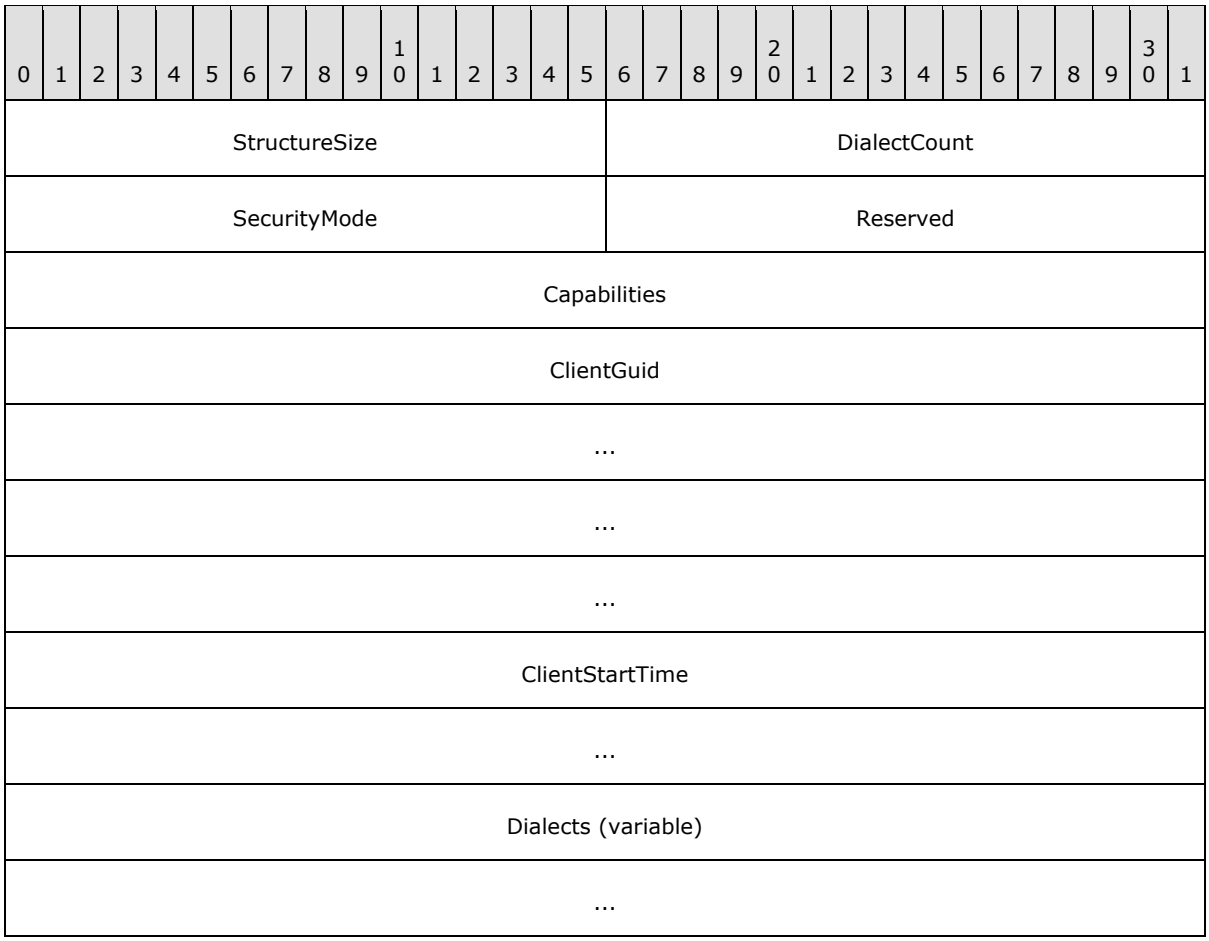
Flags (4 bytes): The server SHOULD set this field to zero. The client MUST ignore it. <3>

PathBuffer (variable): A buffer that contains the Unicode strings for the substitute name and the print name, as described by **SubstituteNameOffset**, **SubstituteNameLength**, **PrintNameOffset**, and **PrintNameLength**. The substitute name string MUST be a Unicode path to the target of the symbolic link. The print name string MUST be a Unicode string, suitable for display to a user, that also identifies the target of the symbolic link.

- For an absolute target that is on a remote machine, the server MUST return the path in the format \??\UNC\server\share\... where server is replaced by the target server name, share is replaced by the target share name, and ... is replaced by the remainder of the path to the target.
- The server SHOULD NOT return symbolic link information with an absolute target that is a local resource, because local evaluation will vary based on client OS. <4>
- For a relative target, the server MUST return a path that does not start with \. The path MUST be evaluated relative to the target of the failed create operation. The path can contain either "." to refer to the current directory or ".." to refer to the parent directory, and may contain multiple elements.

2.2.3 SMB2 NEGOTIATE Request

The SMB2 NEGOTIATE Request packet is used by the client to notify the server what dialects of the SMB 2.0 Protocol the client understands. This request is composed of an SMB 2.0 Protocol header, as specified in section 2.2.1, followed by this request structure:



StructureSize (2 bytes): The client MUST set this field to 36, indicating the size of a NEGOTIATE request. This is not the size of the structure with a single dialect in the **Dialects**[] array. This value MUST be set regardless of the number of dialects sent.

DialectCount (2 bytes): The number of dialects that are contained in the **Dialects**[] array. This value MUST be greater than 0.

SecurityMode (2 bytes): The security mode field MUST be constructed using the following values:

Value	Meaning
SMB2_NEGOTIATE_SIGNING_ENABLED 0x0001	When set, indicates that security signatures are enabled on the client.
SMB2_NEGOTIATE_SIGNING_REQUIRED 0x0002	When set, indicates that security signatures are required by the client.

Reserved (2 bytes): Unused at the present, and MUST be treated as reserved. The client MUST set this to 0, and the server MUST ignore it on receipt.

Capabilities (4 bytes): Specifies protocol capabilities for the client. This field MUST be constructed using the following values:

Value	Meaning
SMB2_GLOBAL_CAP_DFS 0x00000001	When set, indicates that the client supports DFS.

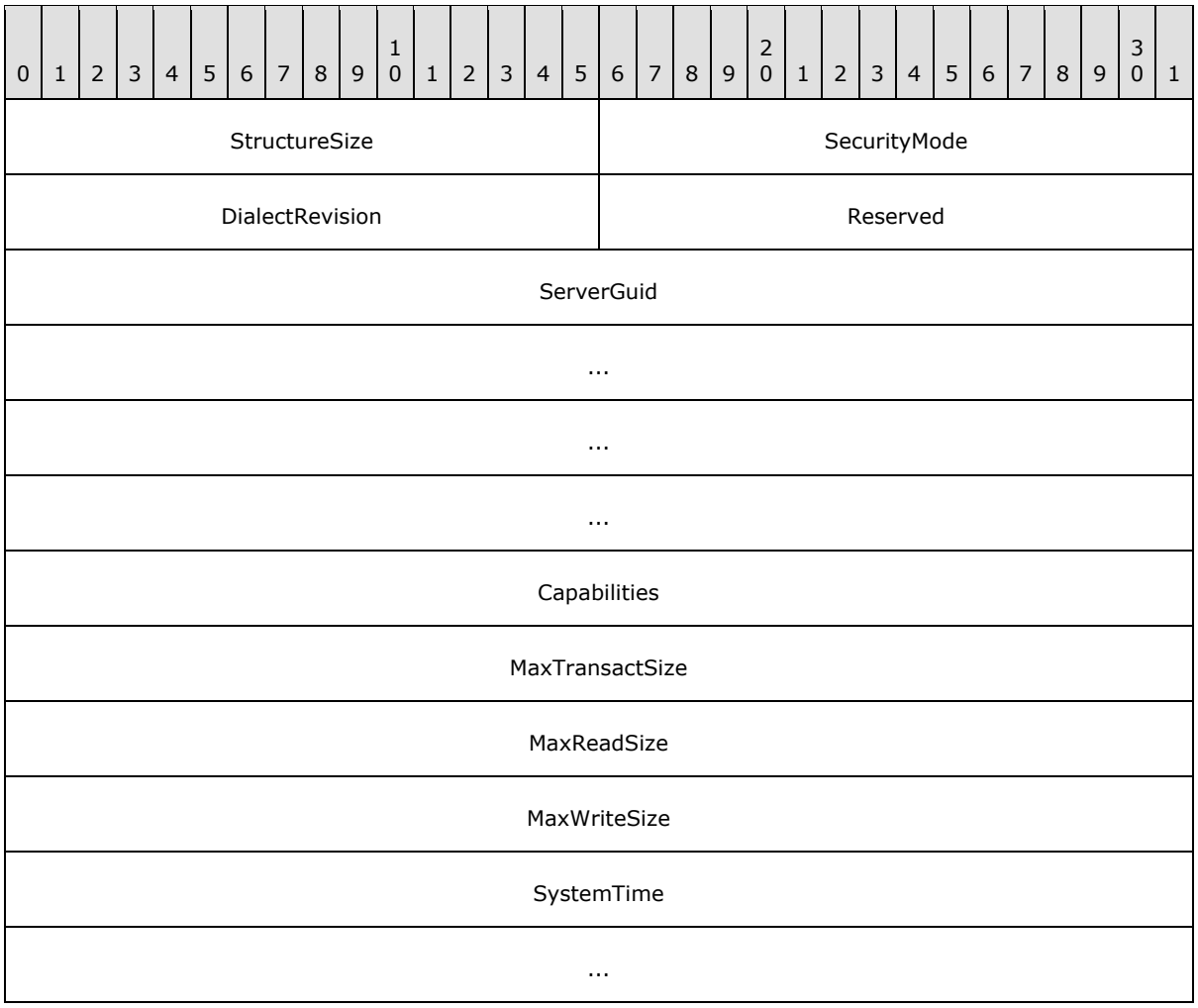
ClientGuid (16 bytes): Unused at the present, and MUST be treated as reserved. The client MUST set this to 0, and the server MUST ignore it on receipt.

ClientStartTime (8 bytes): Unused at the present, and MUST be treated as reserved. The client MUST set this to 0, and the server MUST ignore it on receipt.

Dialects (variable): An array of one or more supported dialect revision numbers. The array MUST contain at least one element of value 0x0202.<5>

2.2.4 SMB2 NEGOTIATE Response

The SMB2 NEGOTIATE Response packet is sent by the server to notify the client of the preferred common dialect. This response is composed of an SMB2 header, as specified in section 2.2.1, followed by this response structure:



ServerStartTime	
...	
SecurityBufferOffset	SecurityBufferLength
Reserved2	
Buffer (variable)	
...	

StructureSize (2 bytes): The server MUST set this field to 65, indicating the size of the response structure, not including the header. The server MUST set it to this value, regardless of how long **Buffer**[] actually is in the response being sent.

SecurityMode (2 bytes): The security mode field MUST be constructed using the following values:

Value	Meaning
SMB2_NEGOTIATE_SIGNING_ENABLED 0x0001	When set, indicates that security signatures are enabled on the server.
SMB2_NEGOTIATE_SIGNING_REQUIRED 0x0002	When set, indicates that security signatures are required by the server.

DialectRevision (2 bytes): The preferred common SMB 2.0 Protocol dialect number from the Dialects array that is sent in the SMB2 NEGOTIATE request (section 2.2.3). The server SHOULD set this field to 0x0202.<6>

Reserved (2 bytes): Unused at the present and MUST be treated as reserved. The server MUST set this to 0.

ServerGuid (16 bytes): A **globally unique identifier** that is generated by the server to uniquely identify this server. This field MUST NOT be used by a client as a secure method of identifying a server. <7>

Capabilities (4 bytes): Specifies protocol capabilities for the server. This field MUST be constructed using the following values:

Value	Meaning
SMB2_GLOBAL_CAP_DFS 0x00000001	When set, indicates that the server supports the Distributed File System (DFS).

MaxTransactSize (4 bytes): The maximum size, in bytes, of the buffer that can be used for QUERY_INFO, QUERY_DIRECTORY, SET_INFO and CHANGE_NOTIFY operations. This field is applicable only for buffers sent by the client in SET_INFO requests, or returned from the server in QUERY_INFO, QUERY_DIRECTORY, and CHANGE_NOTIFY responses. <8>

MaxReadSize (4 bytes): The maximum size, in bytes, of the Length in an SMB2 READ Request (section 2.2.19) that the server will accept.

MaxWriteSize (4 bytes): The maximum size, in bytes, of the Length in an SMB2 WRITE Request (section 2.2.21) that the server will accept.

SystemTime (8 bytes): The system time of the SMB2 server when the SMB2 NEGOTIATE Request was processed; in **FILETIME** format as specified in [MS-DTYP] section 2.3.1.

ServerStartTime (8 bytes): The SMB2 server start time, in **FILETIME** format as specified in [MS-DTYP] section 2.3.1.

SecurityBufferOffset (2 bytes): The offset, in bytes, from the beginning of the SMB 2.0 Protocol header to the security buffer.

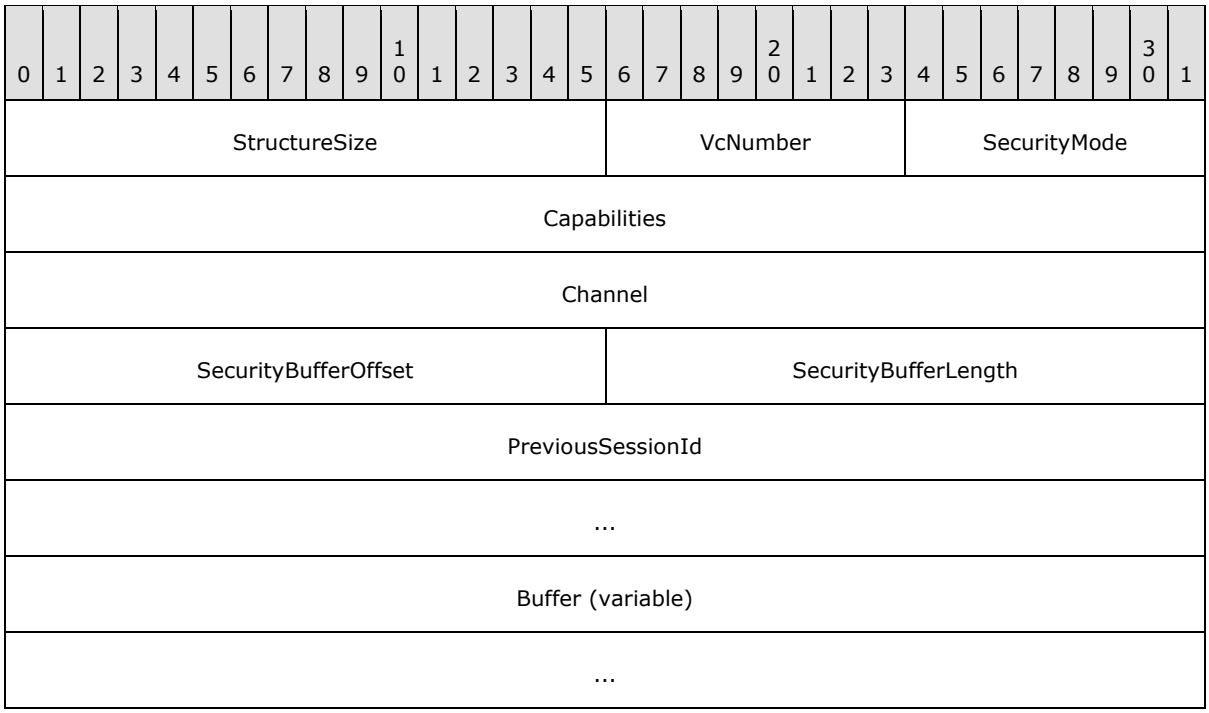
SecurityBufferLength (2 bytes): The length, in bytes, of the security buffer.

Reserved2 (4 bytes): Unused at the present and MUST be treated as reserved. The server MAY set this to any value, and the client MUST ignore it on receipt.

Buffer (variable): The variable-length buffer that contains the security buffer for the response, as specified by **SecurityBufferOffset** and **SecurityBufferLength**. The buffer MUST contain a token as produced by the GSS protocol as specified in section 3.3.5.3.

2.2.5 SMB2 SESSION_SETUP Request

The SMB2 SESSION_SETUP Request packet is sent by the client to request a new authenticated session within a new or existing SMB 2.0 Protocol transport connection to the server. This request is composed of an SMB 2.0 Protocol header as specified in section 2.2.1 followed by this request structure:



StructureSize (2 bytes): The client MUST set this field to 25, indicating the size of the request structure, not including the header. The client MUST set it to this value regardless of how long **Buffer**[] actually is in the request being sent.

VcNumber (1 byte): The number of other transport connections that are already established. The client MUST set this field to 0 regardless of the number of outstanding connections.

SecurityMode (1 byte): The security mode field MUST be constructed using the following values:

Value	Meaning
SMB2_NEGOTIATE_SIGNING_ENABLED 0x01	When set, indicates that security signatures are enabled on the client.
SMB2_NEGOTIATE_SIGNING_REQUIRED 0x02	When set, indicates that security signatures are required by the client.

Capabilities (4 bytes): Specifies protocol capabilities for the client. This field MUST be constructed using the following values:

Value	Meaning
SMB2_GLOBAL_CAP_DFS 0x00000001	When set, indicates that the client supports the Distributed File System (DFS).
SMB2_GLOBAL_CAP_UNUSED1 0x00000002	SHOULD be set to zero and server MUST ignore.<9>
SMB2_GLOBAL_CAP_UNUSED2	SHOULD be set to zero and server MUST ignore.<10>

Value	Meaning
0x00000004	
SMB2_GLOBAL_CAP_UNUSED3 0x00000008	SHOULD be set to zero and server MUST ignore.<11>

Values other than those that are defined in the previous table are unused at present and MUST be treated as reserved. The client MUST set these to zero and the server MUST ignore them on receipt.

Channel (4 bytes): Unused at the present and MUST be treated as reserved. The client MUST set this to 0, and the server MUST ignore it on receipt.

SecurityBufferOffset (2 bytes): The offset, in bytes, from the beginning of the SMB 2.0 Protocol header to the security buffer.

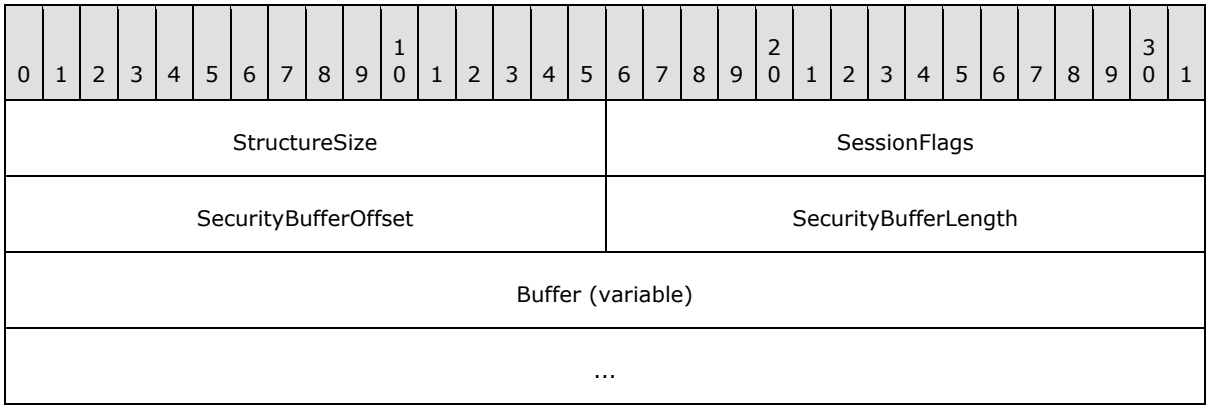
SecurityBufferLength (2 bytes): The length, in bytes, of the security buffer.

PreviousSessionId (8 bytes): A previously established session identifier. If this is a reconnect, the client MUST set this value to its previous session identifier to allow the server to reconnect. If this is not a reconnect, the client MUST set this to 0.

Buffer (variable): A variable-length buffer that contains the security buffer for the request, as specified by **SecurityBufferOffset** and **SecurityBufferLength**. The buffer MUST contain a token as produced by the GSS protocol as specified in section 3.3.5.5.

2.2.6 SMB2 SESSION_SETUP Response

The SMB2 SESSION_SETUP Response packet is sent by the server in response to an SMB2 SESSION_SETUP Request packet. This response is composed of an SMB 2.0 Protocol header, as specified in section 2.2.1, that is followed by this response structure:



StructureSize (2 bytes): The server MUST set this to 9, indicating the size of the fixed part of the response structure not including the header. The server MUST set it to this value regardless of how long **Buffer**[] actually is in the response.

SessionFlags (2 bytes): A flags field that indicates additional information about the session. This field MUST be constructed using the following values:

Value	Meaning
SMB2_SESSION_FLAG_IS_GUEST 0x0001	If set, the client has been authenticated as a guest user.
SMB2_SESSION_FLAG_IS_NULL 0x0002	If set, the client has been authenticated as an anonymous user.

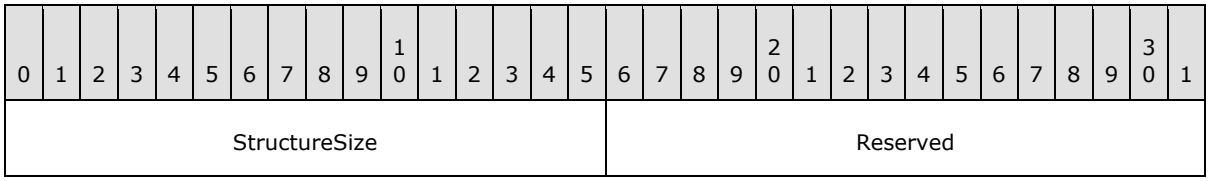
SecurityBufferOffset (2 bytes): The offset, in bytes, from the beginning of the SMB 2.0 Protocol header to the security buffer.

SecurityBufferLength (2 bytes): The length, in bytes, of the security buffer.

Buffer (variable): A variable-length buffer that contains the security buffer for the response, as specified by **SecurityBufferOffset** and **SecurityBufferLength**. The buffer MUST contain a token as produced by the GSS protocol as specified in section 3.2.5.3.

2.2.7 SMB2 LOGOFF Request

The SMB2 LOGOFF Request packet.

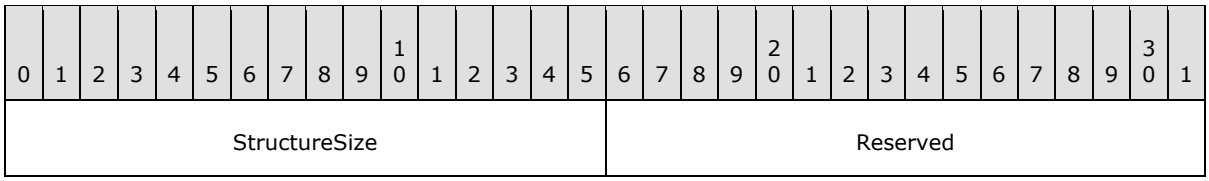


StructureSize (2 bytes): The client MUST set this field to 4, indicating the size of the request structure not including the header.

Reserved (2 bytes): Unused at the present and MUST be treated as reserved. The client MUST set this to 0, and the server MUST ignore it on receipt.

2.2.8 SMB2 LOGOFF Response

The SMB2 LOGOFF Response packet is sent by the server to confirm that an SMB2 LOGOFF Request (section 2.2.7) was completed successfully. This response is composed of an SMB2 header, as specified in section 2.2.1, followed by this request structure:

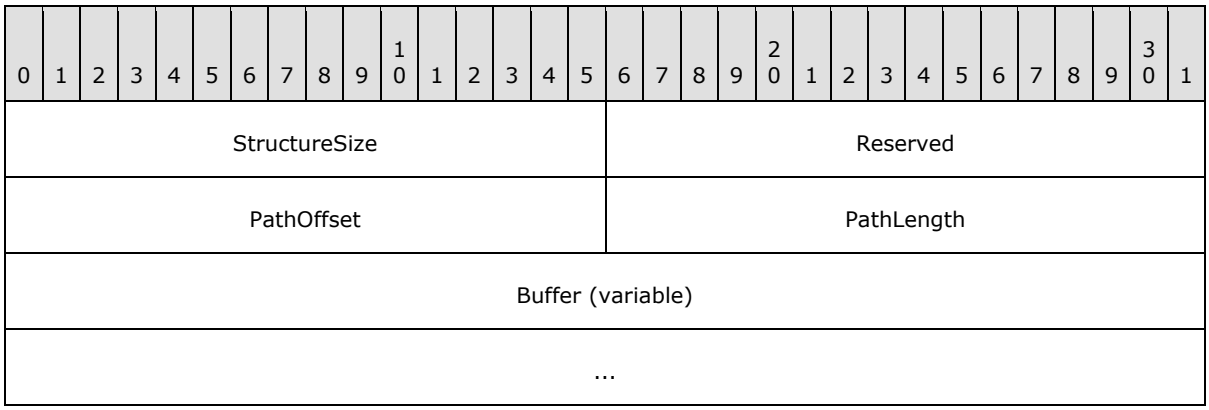


StructureSize (2 bytes): The server MUST set this field to 4, indicating the size of the response structure, not including the header.

Reserved (2 bytes): Unused at the present and MUST be treated as reserved. The server MUST set this to 0, and the client MUST ignore it on receipt.

2.2.9 SMB2 TREE_CONNECT Request

The SMB2 TREE_CONNECT Request packet is sent by a client to request access to a particular share on the server. This request is composed of an SMB2 Packet Header (section 2.2.1) that is followed by this request structure:



StructureSize (2 bytes): The client MUST set this field to 9, indicating the size of the request structure, not including the header. The client MUST set it to this value regardless of how long **Buffer**[] actually is in the request being sent.

Reserved (2 bytes): Unused at the present and MUST be treated as reserved. The client MUST set this to 0, and the server MUST ignore it on receipt.

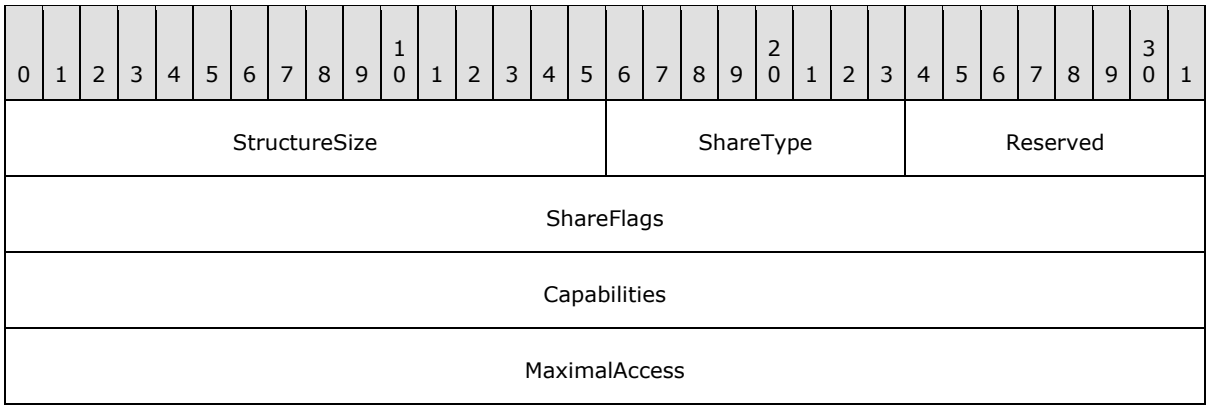
PathOffset (2 bytes): The offset, in bytes, of the full share path name from the beginning of the packet header.

PathLength (2 bytes): The length, in bytes, of the path name.

Buffer (variable): A variable-length buffer that contains the path name of the share in Unicode in the form "\\server\share" for the request, as described by **PathOffset** and **PathLength**. The server component of the path MUST be a NetBIOS name, a DNS name, or a textual IPv4 or IPv6 address, and it MUST be fewer than 256 characters in length. The share component of the path MUST be less than or equal to 80 characters in length.<12>

2.2.10 SMB2 TREE_CONNECT Response

The SMB2 TREE_CONNECT Response packet is sent by the server when an SMB2 TREE_CONNECT request is processed successfully by the server. The server MUST set the **TreeId** of the newly created tree connect in the SMB 2.0 Protocol header of the response. This response is composed of an SMB2 Packet Header (section 2.2.1) that is followed by this response structure:



StructureSize (2 bytes): The server MUST set this field to 16, indicating the size of the response structure, not including the header.

ShareType (1 byte): The type of share being accessed. This field MUST contain one of the following values:

Value	Meaning
SMB2_SHARE_TYPE_DISK 0x01	Physical disk share.
SMB2_SHARE_TYPE_PIPE 0x02	Named pipe share.
SMB2_SHARE_TYPE_PRINT 0x03	Printer share.

Reserved (1 byte): Unused at the present, and MUST be treated as reserved. The server MUST set this to 0, and the client MUST ignore it on receipt.

ShareFlags (4 bytes): This field contains properties for this share. This field MUST contain one of the following offline caching properties: SMB2_SHAREFLAG_MANUAL_CACHING, SMB2_SHAREFLAG_AUTO_CACHING, SMB2_SHAREFLAG_VDO_CACHING and SMB2_SHAREFLAG_NO_CACHING. For more information about offline caching, see [OFFLINE]. This field MAY contain zero or more of the following values: SHI1005_FLAGS_DFS, SHI1005_FLAGS_DFS_ROOT, SHI1005_FLAGS_RESTRICT_EXCLUSIVE_OPENS, SHI1005_FLAGS_FORCE_SHARED_DELETE, SHI1005_FLAGS_ALLOW_NAMESPACE_CACHING, SHI1005_FLAGS_ACCESS_BASED_DIRECTORY_ENUM.

Value	Meaning
SMB2_SHAREFLAG_MANUAL_CACHING 0x00000000	The client MAY cache files that are explicitly selected by the user for offline use.
SMB2_SHAREFLAG_AUTO_CACHING 0x00000010	The client MAY automatically cache files that are used by the user for offline access.
SMB2_SHAREFLAG_VDO_CACHING 0x00000020	The client MAY automatically cache files that are used by the user for offline access, and MAY use those files in an offline mode

Value	Meaning
	even if the share is available.
SMB2_SHAREFLAG_NO_CACHING 0x00000030	Offline caching MUST NOT occur.
SHI1005_FLAGS_DFS 0x00000001	The specified share is present in a DFS tree structure.
SHI1005_FLAGS_DFS_ROOT 0x00000002	The specified share is the root volume in a DFS tree structure.
SHI1005_FLAGS_RESTRICT_EXCLUSIVE_OPENS 0x00000100	The specified share disallows exclusive file opens that deny reads to an open file.
SHI1005_FLAGS_FORCE_SHARED_DELETE 0x00000200	Shared files in the specified share can be forcibly deleted.
SHI1005_FLAGS_ALLOW_NAMESPACE_CACHING 0x00000400	Clients are allowed to cache the namespace of the specified share.
SHI1005_FLAGS_ACCESS_BASED_DIRECTORY_ENUM 0x00000800	The server will filter directory entries based on the access permissions of the client.

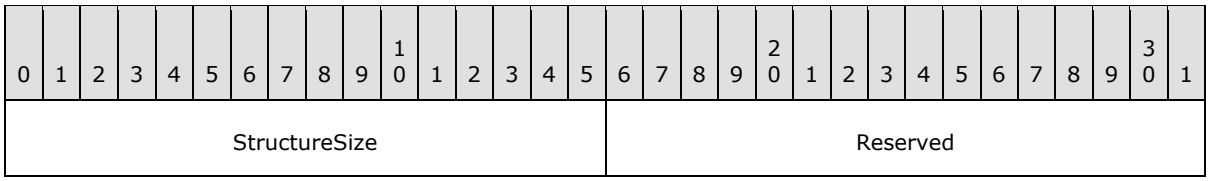
Capabilities (4 bytes): Indicates various capabilities for this share. This field MUST be constructed using the following values:

Value	Meaning
SMB2_GLOBAL_CAP_DFS 0x00000008	When set, indicates that the server supports the Distributed File System (DFS).

MaximalAccess (4 bytes): Contains the maximal access for the user that establishes the tree connect on the share based on the share's permissions. This value takes the form as specified in section 2.2.13.1.

2.2.11 SMB2 TREE_DISCONNECT Request

The SMB2 TREE_DISCONNECT Request (section 2.2.11) packet is sent by the client to request that the tree connect that is specified in the **TreeId** within the SMB2 header be disconnected. This request is composed of an SMB2 header, as specified in section 2.2.1, that is followed by this variable-length request structure:

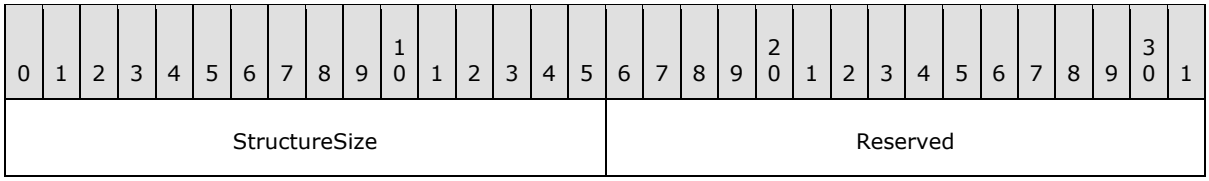


StructureSize (2 bytes): The client MUST set this field to 4, indicating the size of the request structure, not including the header.

Reserved (2 bytes): Unused at the present and MUST be treated as reserved. The client MUST set this to 0, and the server MUST ignore it on receipt.

2.2.12 SMB2 TREE_DISCONNECT Response

The SMB2 TREE_DISCONNECT Response (section 2.2.12) packet is sent by the server to confirm that an SMB2 TREE_DISCONNECT Request (section 2.2.11) was successfully processed. This response is composed of an SMB2 header, as specified in section 2.2.1, that is followed by this request structure:



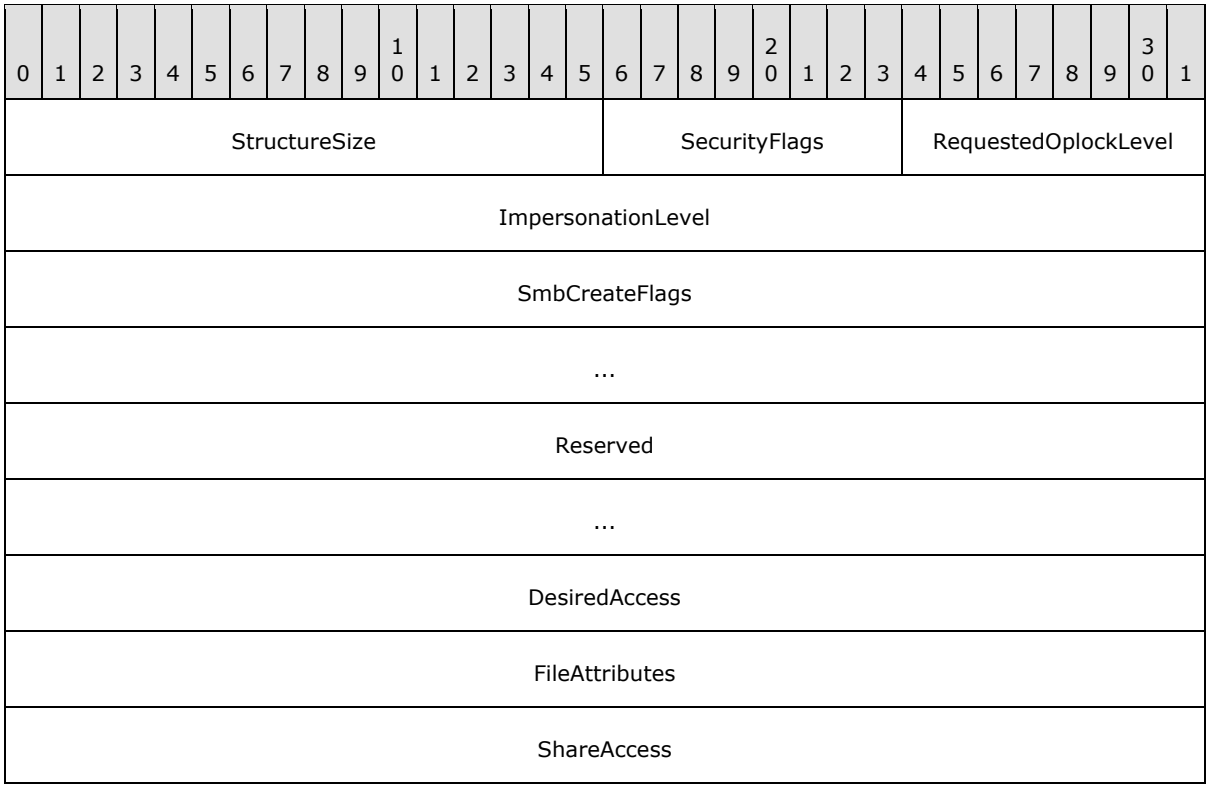
StructureSize (2 bytes): The server MUST set this field to 4, indicating the size of the response structure, not including the header.

Reserved (2 bytes): Unused at the present and MUST be treated as reserved. The server MUST set this to 0, and the client MUST ignore it on receipt.

2.2.13 SMB2 CREATE Request

The SMB2 CREATE Request packet is sent by a client to request either creation of or access to a file. In case of a named pipe or printer, the server MUST create a new file.

This request is composed of an SMB2 Packet Header, as specified in section 2.2.1, that is followed by this request structure:



CreateDisposition	
CreateOptions	
NameOffset	NameLength
CreateContextsOffset	
CreateContextsLength	
Buffer (variable)	
...	

StructureSize (2 bytes): The client MUST set this field to 57, indicating the size of the request structure, not including the header. The client MUST set it to this value regardless of how long **Buffer**[] actually is in the request being sent.

SecurityFlags (1 byte): Unused at the present, and MUST be treated as reserved. The sender MUST set this to 0, and the receiver MUST ignore it.

RequestedOplockLevel (1 byte): The requested **oplock** level. This field MUST contain one of the following values.<13> For named pipes, the server MUST always revert to SMB2_OPLOCK_LEVEL_NONE irrespective of the value of this field.

Value	Meaning
SMB2_OPLOCK_LEVEL_NONE 0x00	No oplock is requested.
SMB2_OPLOCK_LEVEL_II 0x01	A level II oplock is requested.
SMB2_OPLOCK_LEVEL_EXCLUSIVE 0x08	An exclusive oplock is requested.
SMB2_OPLOCK_LEVEL_BATCH 0x09	A batch oplock is requested.

ImpersonationLevel (4 bytes): This field specifies the impersonation level of the application that is issuing the create request. This field MUST contain one of the following values:

Value	Meaning
Anonymous 0x00000000	The client is anonymous to the server. The server process SHOULD impersonate the client, but the impersonation token MUST NOT contain any information about the client. The server process MUST NOT obtain identification information about the client.
Identification 0x00000001	The server SHOULD obtain the client's identity, and the server SHOULD impersonate the client to perform access control list (ACL) checks.

Value	Meaning
Impersonation 0x00000002	The server SHOULD impersonate the client's security context while acting on behalf of the client. The server MAY access local resources as the client.
Delegate 0x00000003	The server SHOULD impersonate the client's security context while acting on behalf of the client. During impersonation, the client's credentials (both local and network) MAY be passed to any number of machines.

Impersonation is specified in [MS-SECO] section 5; for more information about impersonation, see [MSDN-IMPERS].

SmbCreateFlags (8 bytes): Unused at the present, and MUST be treated as reserved. The client sets this to any value, and the server MUST ignore it on receipt.

Reserved (8 bytes): Unused at the present, and MUST be treated as reserved. The client sets this to any value, and the server MUST ignore it on receipt.

DesiredAccess (4 bytes): The level of access that is wanted, as specified in section 2.2.13.1. If DesiredAccess values other than FILE_WRITE_DATA, FILE_APPEND_DATA or GENERIC_WRITE are set for a printer file, a STATUS_NOT_SUPPORTED will be returned.

FileAttributes (4 bytes): This field MUST be a combination of the values specified in [MS-FSCC] section 2.6. This field is only valid when a file is created. The server MUST ignore the FileAttributes field when opening an existing file, and the client SHOULD set the FileAttributes field to 0. <14> For a printer, all attributes except FILE_ATTRIBUTE_DIRECTORY are valid. For print files, the server MUST fail the open if FileAttributes includes FILE_ATTRIBUTE_DIRECTORY with the error code STATUS_NOT_SUPPORTED. The server MUST ignore FileAttributes for pipe opens. All other attributes MUST be ignored by the server.

ShareAccess (4 bytes): Specifies the sharing mode for the open. If ShareAccess values of FILE_SHARE_READ, FILE_SHARE_WRITE and FILE_SHARE_DELETE are set for a printer file or a named pipe, the server SHOULD ignore these values<15>This field MUST be constructed using the following values:

Value	Meaning
FILE_SHARE_READ 0x00000001	When set, indicates that other opens are allowed to read this file while this open is present. This bit MUST NOT be set for a named pipe or a printer file. Each open creates a new instance of a named pipe. Likewise, opening a printer file always creates a new file.
FILE_SHARE_WRITE 0x00000002	When set, indicates that other opens are allowed to write this file while this open is present. This bit MUST NOT be set for a named pipe or a printer file. Each open creates a new instance of a named pipe. Likewise, opening a printer file always creates a new file.
FILE_SHARE_DELETE 0x00000004	When set, indicates that other opens are allowed to delete or rename this file while this open is present. This bit MUST NOT be set for a named pipe or a printer file. Each open creates a new instance of a named pipe. Likewise, opening a printer file always creates a new file.

CreateDisposition (4 bytes): Defines the action the server MUST take if the file that is specified in the name field already exists. For opening named pipes, this field MAY be set to any value by the client and MUST be ignored by the server. For CreateDisposition values not permitted for Printer objects, the server MUST fail the request with

STATUS_OBJECT_NAME_NOT_FOUND. For other files, this field MUST contain one of the following values:

Value	Meaning
FILE_SUPERSEDE 0x00000000	If the file already exists, supersede it. Otherwise, create the file. This value SHOULD NOT be used for a printer object. <16>
FILE_OPEN 0x00000001	If the file already exists, return success; otherwise, fail the operation. MUST NOT be used for a printer object.
FILE_CREATE 0x00000002	If the file already exists, fail the operation; otherwise, create the file.
FILE_OPEN_IF 0x00000003	Open the file if it already exists; otherwise, create the file. This value SHOULD NOT be used for a printer object. <17>
FILE_OVERWRITE 0x00000004	Overwrite the file if it already exists; otherwise, fail the operation. MUST NOT be used for a printer object.
FILE_OVERWRITE_IF 0x00000005	Overwrite the file if it already exists; otherwise, create the file. This value SHOULD NOT be used for a printer object. <18>

CreateOptions (4 bytes): Specifies the options to be applied when creating or opening the file. Combinations of the bit positions listed below are valid, unless otherwise noted. This field MUST be constructed using the following values:

Value	Meaning
FILE_DIRECTORY_FILE 0x00000001	The file being created or opened is a directory file. With this flag, the **CreateDisposition** field MUST be set to FILE_CREATE or FILE_OPEN_IF. With this flag, only the following CreateOptions values are valid: FILE_WRITE_THROUGH, and FILE_OPEN_FOR_BACKUP_INTENT.
FILE_WRITE_THROUGH 0x00000002	The server MUST propagate writes to this open to persistent storage before returning success to the client on write operations.
FILE_SEQUENTIAL_ONLY 0x00000004	A hint indicating that accesses to the file will be sequential. This flag value is incompatible with the FILE_RANDOM_ACCESS value, which indicates that the accesses to the file can be random.
FILE_NO_INTERMEDIATE_BUFFERING 0x00000008	The server or underlying object store SHOULD NOT cache data at intermediate layers and SHOULD allow it to flow through to persistent storage.
FILE_NON_DIRECTORY_FILE 0x00000040	The file being opened MUST NOT be a directory file or this call MUST be failed. This flag MUST NOT be used with FILE_DIRECTORY_FILE.
FILE_NO_EA_KNOWLEDGE 0x00000200	The caller does not understand how to handle extended attributes. If extended attributes are associated with the file being opened, the server MUST fail this request.
FILE_OPEN_FOR_BACKUP_INTENT	The file is being opened for backup intent. That is, it is being opened or created for the purposes of either a

Value	Meaning
0x00004000	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_RANDOM_ACCESS 0x00000800	A hint that indicates that accesses to the file can be random; so sequential read-ahead operations SHOULD NOT be performed on the file. This flag value is incompatible with the FILE_SEQUENTIAL_ONLY value, which indicates that the accesses to the file will be sequential.
FILE_NO_COMPRESSION 0x00008000	The file cannot be compressed.
FILE_OPEN_REPARSE_POINT 0x00200000	If the file or directory being opened is a reparse point, open the reparse point itself rather than the target that the reparse point references.
FILE_OPEN_NO_RECALL 0x00400000	In an HSM (Hierarchical Storage Management) environment, this flag means the file should not be recalled from tertiary storage such as tape. The recall can take several minutes. The caller can specify this flag to avoid those delays.

NameOffset (2 bytes): The offset, in bytes, from the beginning of the SMB2 header to the 8-byte aligned file name. The file name is relative to the share that is found by the **TreeId** in the SMB2 header. If no name is provided, indicating an open of the root directory of the share, this field MUST be 0.

NameLength (2 bytes): The length of the file name, in bytes. Each individual file name component MUST NOT exceed 255 Unicode characters and the full path name MUST NOT exceed 32760 Unicode characters. If no file name is provided, this field MUST be set to 0.

CreateContextsOffset (4 bytes): The offset, in bytes, from the beginning of the SMB2 header to the first 8-byte aligned SMB2_CREATE_CONTEXT structure in the request. If no SMB2_CREATE_CONTEXTs are being sent, this value MUST be 0.

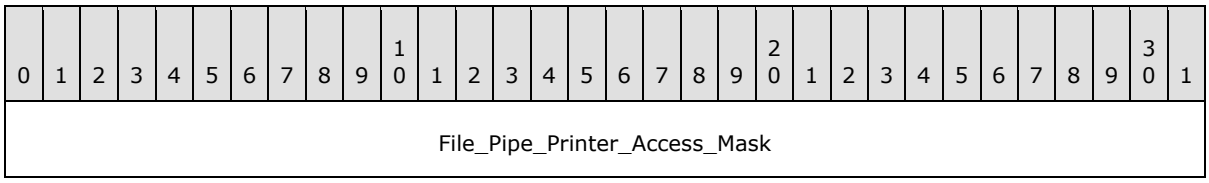
CreateContextsLength (4 bytes): The length, in bytes, of the list of SMB2_CREATE_CONTEXT structures sent in this request.

Buffer (variable): A variable-length buffer that contains the Unicode file name and **create context** list, as defined by **NameOffset**, **NameLength**, **CreateContextsOffset**, and **CreateContextsLength**.

2.2.13.1 SMB2 Access Mask Encoding

The SMB2 Access Mask Encoding in SMB2 is a 4-byte bit field value that contains an array of flags. An access mask can specify access for one of two basic groups: either for a file, pipe, or printer (specified in section 2.2.13.1.1) or for a directory (specified in section 2.2.13.1.2). Each access mask MUST be a combination of zero or more of the bit positions that are shown below.

2.2.13.1.1 File_Pipe_Printer_Access_Mask

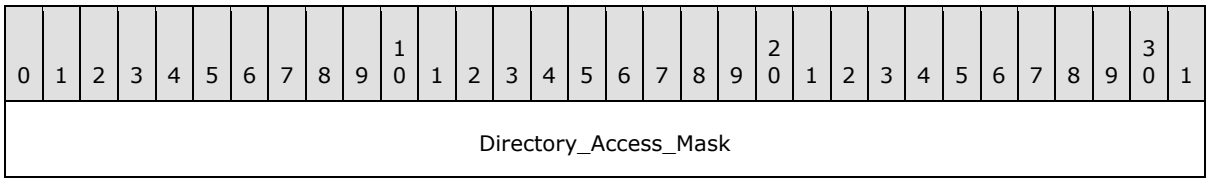


File_Pipe_Printer_Access_Mask (4 bytes): For a file, pipe, or printer, the value MUST be constructed using the following values (for a printer, the value MUST have at least one of the following: FILE_WRITE_DATA, FILE_APPEND_DATA, or GENERIC_WRITE).

Value	Meaning
FILE_READ_DATA 0x00000001	This value indicates the right to read data from the file or named pipe.
FILE_WRITE_DATA 0x00000002	This value indicates the right to write data into the file or named pipe beyond the end of the file.
FILE_APPEND_DATA 0x00000004	This value indicates the right to append data into the file or named pipe.
FILE_READ_EA 0x00000008	This value indicates the right to read the extended attributes of the file or named pipe.
FILE_WRITE_EA 0x00000010	This value indicates the right to write or change the extended attributes to the file or named pipe.
FILE_EXECUTE 0x00000020	This value indicates the right to execute the file.
FILE_READ_ATTRIBUTES 0x00000080	This value indicates the right to read the attributes of the file.
FILE_WRITE_ATTRIBUTES 0x00000100	This value indicates the right to change the attributes of the file.
DELETE 0x00010000	This value indicates the right to delete the file.
READ_CONTROL 0x00020000	This value indicates the right to read the security descriptor for the file or named pipe.
WRITE_DAC 0x00040000	This value indicates the right to change the **discretionary access control list (DACL)** in the security descriptor for the file or named pipe. For the DACL data structure, see **ACL** in [MS-DTYP].
WRITE_OWNER 0x00080000	This value indicates the right to change the owner in the security descriptor for the file or named pipe.
SYNCHRONIZE 0x00100000	This value SHOULD NOT be used by the sender and MUST be ignored by the receiver.
ACCESS_SYSTEM_SECURITY 0x01000000	This value indicates the right to read or change the **system access control list (SACL)** in the security descriptor for the file or named pipe. For the SACL data structure, see **ACL** in [MS-DTYP].
MAXIMAL_ACCESS	This value indicates that the client is requesting an open to the file

Value	Meaning
0x02000000	with the highest level of access the client has on this file. If no access is granted for the client on this file, the server MUST fail the open with STATUS_ACCESS_DENIED.
GENERIC_ALL 0x10000000	This value indicates a request for all the access flags that are previously listed except MAXIMAL_ACCESS and ACCESS_SYSTEM_SECURITY.
GENERIC_EXECUTE 0x20000000	This value indicates a request for the following combination of access flags listed above: FILE_READ_ATTRIBUTES| FILE_EXECUTE| SYNCHRONIZE| READ_CONTROL.
GENERIC_WRITE 0x40000000	This value indicates a request for the following combination of access flags listed above: FILE_WRITE_DATA| FILE_APPEND_DATA| FILE_WRITE_ATTRIBUTES| FILE_WRITE_EA| SYNCHRONIZE| READ_CONTROL.
GENERIC_READ 0x80000000	This value indicates a request for the following combination of access flags listed above: FILE_READ_DATA| FILE_READ_ATTRIBUTES| FILE_READ_EA| SYNCHRONIZE| READ_CONTROL.

2.2.13.1.2 Directory_Access_Mask



Directory_Access_Mask (4 bytes): For a directory, the value MUST be constructed using the following values:

Value	Meaning
FILE_LIST_DIRECTORY 0x00000001	This value indicates the right to enumerate the contents of the directory.
FILE_ADD_FILE 0x00000002	This value indicates the right to create a file under the directory.
FILE_ADD_SUBDIRECTORY 0x00000004	This value indicates the right to add a sub-directory under the directory.
FILE_READ_EA 0x00000008	This value indicates the right to read the extended attributes of the directory.
FILE_WRITE_EA 0x00000010	This value indicates the right to write or change the extended attributes of the directory.
FILE_TRAVERSE 0x00000020	This value indicates the right to traverse this directory if the server enforces traversal checking.
FILE_DELETE_CHILD 0x00000040	This value indicates the right to delete the files and directories within this directory.

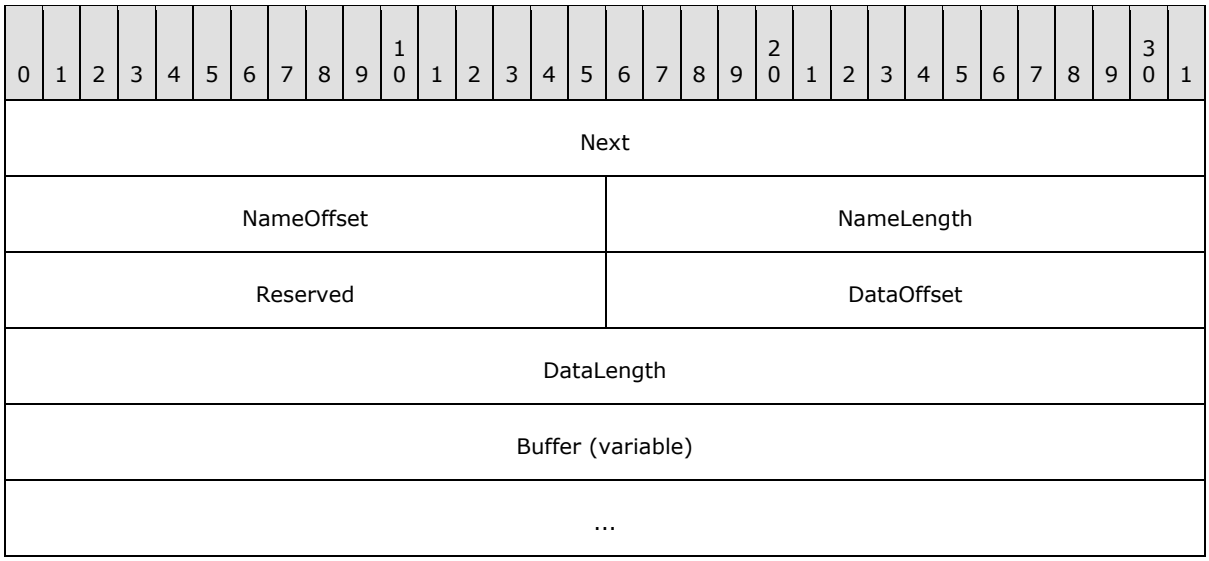
Value	Meaning
FILE_READ_ATTRIBUTES 0x00000080	This value indicates the right to read the attributes of the directory.
FILE_WRITE_ATTRIBUTES 0x00000100	This value indicates the right to change the attributes of the directory.
DELETE 0x00010000	This value indicates the right to delete the directory.
READ_CONTROL 0x00020000	This value indicates the right to read the security descriptor for the directory.
WRITE_DAC 0x00040000	This value indicates the right to change the DACL in the security descriptor for the directory. For the DACL data structure, see **ACL** in [MS-DTYP].
WRITE_OWNER 0x00080000	This value indicates the right to change the owner in the security descriptor for the directory.
SYNCHRONIZE 0x00100000	This value SHOULD NOT be used by the sender and MUST be ignored by the receiver.
ACCESS_SYSTEM_SECURITY 0x01000000	This value indicates the right to read or change the SACL in the security descriptor for the directory. For the SACL data structure, see **ACL** in [MS-DTYP].
MAXIMAL_ACCESS 0x02000000	This value indicates that the client is requesting an open to the directory with the highest level of access the client has on this directory. If no access is granted for the client on this directory, the server MUST fail the open with STATUS_ACCESS_DENIED.
GENERIC_ALL 0x10000000	This value indicates a request for all the access flags that are listed above except MAXIMAL_ACCESS and ACCESS_SYSTEM_SECURITY.
GENERIC_EXECUTE 0x20000000	This value indicates a request for the following access flags listed above: FILE_READ_ATTRIBUTES| FILE_TRAVERSE| SYNCHRONIZE| READ_CONTROL.
GENERIC_WRITE 0x40000000	This value indicates a request for the following access flags listed above: FILE_ADD_FILE| FILE_ADD_SUBDIRECTORY| FILE_WRITE_ATTRIBUTES| FILE_WRITE_EA| SYNCHRONIZE| READ_CONTROL.
GENERIC_READ 0x80000000	This value indicates a request for the following access flags listed above: FILE_LIST_DIRECTORY| FILE_READ_ATTRIBUTES| FILE_READ_EA| SYNCHRONIZE| READ_CONTROL.

2.2.13.2 SMB2_CREATE_CONTEXT Request Values

The SMB2_CREATE_CONTEXT structure is used by the SMB2 CREATE Request and the SMB2 CREATE Response to encode additional flags and attributes: in requests to specify how the CREATE request MUST be processed, and in responses to specify how the CREATE request was in fact processed.

There is no required ordering when multiple create context structures are used. The server MUST support receiving the contexts in any order.

Each structure takes the following form:



Next (4 bytes): The offset from the beginning of this create context to the beginning of a subsequent 8-byte aligned create context. This field MUST be set to 0 if there are no subsequent contexts.

NameOffset (2 bytes): The offset from the beginning of this structure to its 8-byte aligned name value. The name is represented as four or more ASCII characters and MUST be one of the values provided in the following table. The structure name indicates what information is encoded by the data payload. The following values are the valid create context values. The names are case-sensitive. More details are provided for each of these values in the following subsections.

Value	Meaning
SMB2_CREATE_EA_BUFFER "ExtA"	The data contains the extended attributes that MUST be stored on the created file. This value MUST NOT be set for named pipes and print files.
SMB2_CREATE_SD_BUFFER "SecD"	The data contains a security descriptor that MUST be stored on the created file. This value MUST NOT be set for named pipes and print files.
SMB2_CREATE_DURABLE_HANDLE_REQUEST "DHnQ"	The client is requesting the open to be durable (see section 3.3.5.9.6).
SMB2_CREATE_DURABLE_HANDLE_RECONNECT "DHnC"	The client is requesting to reconnect to a **durable open** after being disconnected (see section 3.3.5.9.7).
SMB2_CREATE_ALLOCATION_SIZE "AlSi"	The data contains the required allocation size of the newly created file.
SMB2_CREATE_QUERY_MAXIMAL_ACCESS "MxAc"	The client is requesting that the server return maximal access information.
SMB2_CREATE_TIMEWARP_TOKEN	The client is requesting that the server open an

Value	Meaning
"TWrp"	earlier version of the file with the provided time stamp.
SMB2_CREATE_QUERY_ON_DISK_ID "QFid"	The client is requesting that the server return a 32-byte opaque blob that uniquely identifies the file being opened on disk. No data is passed to the server by the client.

NameLength (2 bytes): The length, in bytes, of the create context name.

Reserved (2 bytes): Unused at the present, and MUST be treated as reserved. This value MUST be set to 0 by the sender, and ignored by the receiver.

DataOffset (2 bytes): The offset, in bytes, from the beginning of this structure to the 8-byte aligned data payload.

DataLength (4 bytes): The length, in bytes, of the data. The format of the data is determined by the type of SMB2_CREATE_CONTEXT request, as outlined in the following sections. The type is indicated in the **NameOffset** field in the message.

Buffer (variable): A variable-length buffer that contains the name and data fields, as defined by **NameOffset**, **NameLength**, **DataOffset**, and **DataLength**.

2.2.13.2.1 SMB2_CREATE_EA_BUFFER

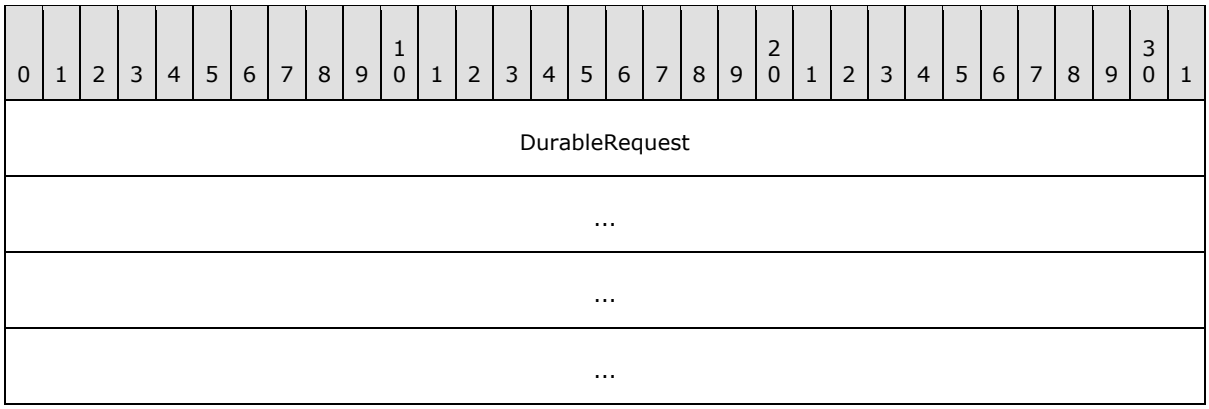
The SMB2_CREATE_EA_BUFFER context is specified on an SMB2 CREATE Request (section 2.2.13) when the client is applying extended attributes as part of creating a new file. This MUST be ignored for requests to open an existing file, and is not applicable for pipes and printers. The extended attributes are provided in the **Data** buffer of the SMB2_CREATE_CONTEXT request and MUST be in the format that is specified for FILE_FULL_EA_INFORMATION in [MS-FSCC] section 2.4.15.

2.2.13.2.2 SMB2_CREATE_SD_BUFFER

The SMB2_CREATE_SD_BUFFER context is specified on an SMB2 CREATE Request when the client is applying a security descriptor to a newly created file. This MUST be ignored for requests to open an existing file, and is not applicable for pipes and printers. The **Data** field of the SMB2_CREATE_CONTEXT MUST contain a security descriptor that MUST be a self-relative **SECURITY_DESCRIPTOR** in the format as specified in [MS-DTYP] section 2.4.6.

2.2.13.2.3 SMB2_CREATE_DURABLE_HANDLE_REQUEST

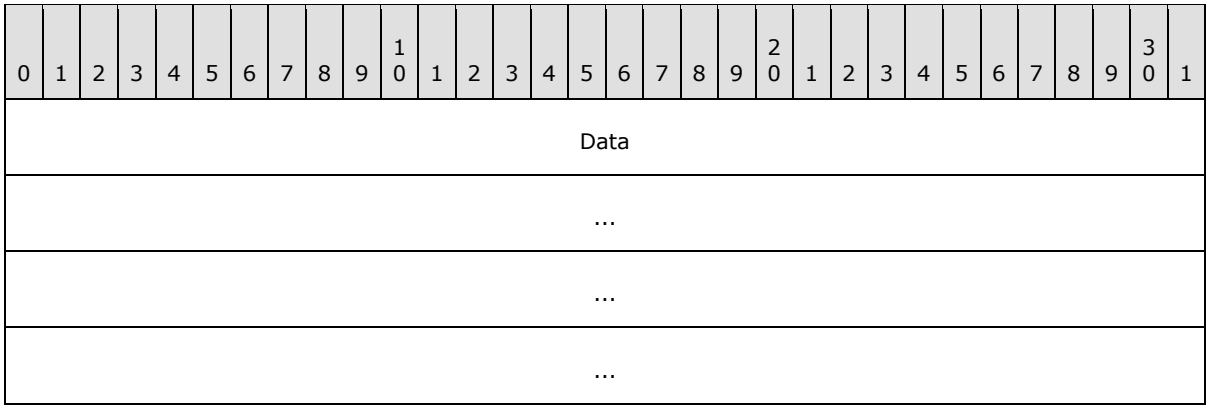
The SMB2_CREATE_DURABLE_HANDLE_REQUEST context is specified on an SMB2 CREATE Request when the client is requesting that the open be durable (that is, that it allow re-establishment after disconnect). It MUST NOT be used for pipes. <19> The format of the **Data** field of this SMB2_CREATE_CONTEXT MUST be as follows:



DurableRequest (16 bytes): A 16-byte field that is currently not used and MUST be treated as reserved for future use. This value MUST be set to 0 by the client and ignored by the server.

2.2.13.2.4 SMB2_CREATE_DURABLE_HANDLE_RECONNECT

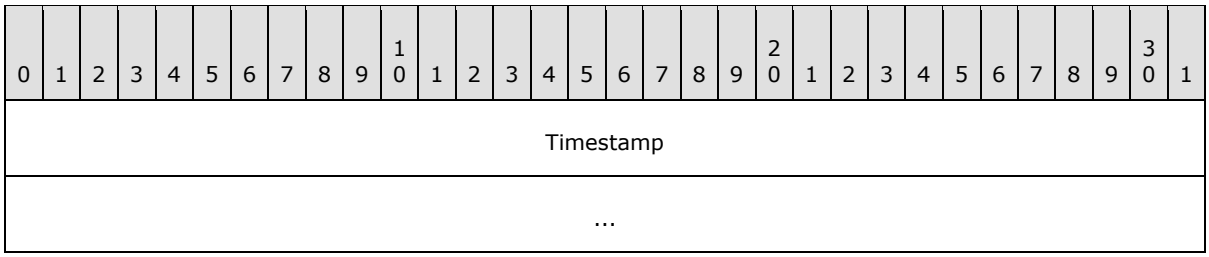
The SMB2 CREATE_DURABLE_HANDLE_RECONNECT context is specified when the client is attempting to reestablish a durable open as specified in section 3.3.5.9.7.



Data (16 bytes): An **SMB2_FILEID** structure, as specified in section 2.2.14.1, for the open that is being reestablished.

2.2.13.2.5 SMB2_CREATE_QUERY_MAXIMAL_ACCESS

The SMB2 CREATE_QUERY_MAXIMAL_ACCESS context is specified on an SMB2 CREATE Request when the client is requesting the server to retrieve maximal access information as part of processing the open. The **Data** field of the SMB2_CREATE_CONTEXT MUST either contain the following structure or be empty (0 bytes in length).

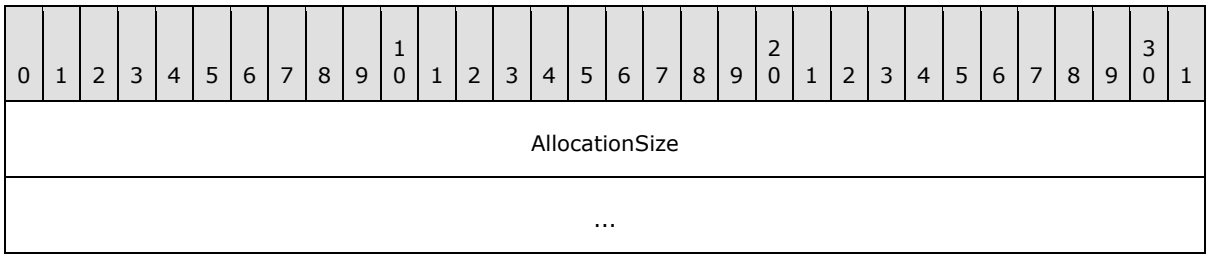


Timestamp (8 bytes): A time stamp in the **FILETIME** format, as specified in [MS-DTYP] section 2.3.1

This time stamp MUST be compared (as specified in section 3.3.5.9.5) with the LastModifiedTime of the file being opened in the underlying object store.

2.2.13.2.6 SMB2_CREATE_ALLOCATION_SIZE

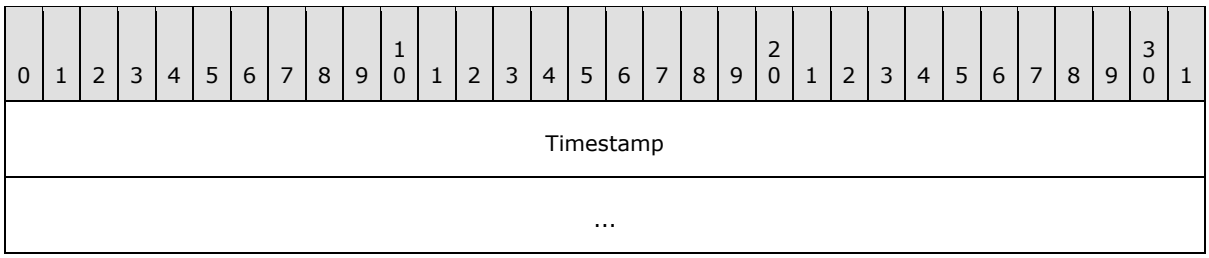
The SMB2 CREATE_ALLOCATION_SIZE context is specified on an SMB2 CREATE Request (section 2.2.13) when the client is setting the initial allocation size of a newly created file. The **Data** field of the SMB2_CREATE_CONTEXT MUST be as follows:



AllocationSize (8 bytes): The size, in bytes, that the newly created file MUST have reserved on disk.

2.2.13.2.7 SMB2_CREATE_TIMEWARP_TOKEN

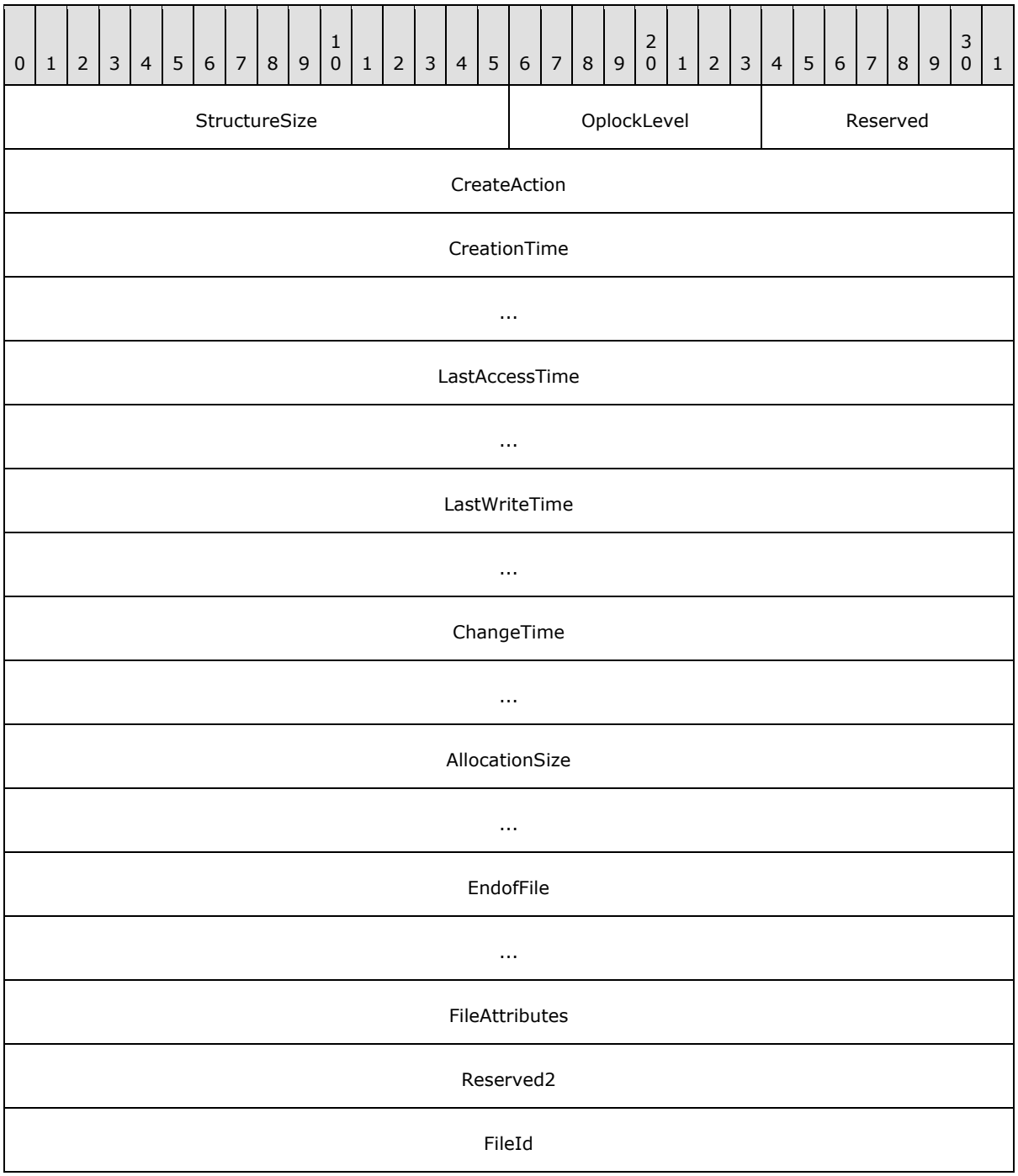
The SMB2_CREATE_TIMEWARP_TOKEN context is specified on an SMB2 CREATE Request (section 2.2.13) when the client is requesting the server to open a version of the file at a previous point in time. The **Data** field of the SMB2_CREATE_CONTEXT MUST contain the following structure:



Timestamp (8 bytes): The time stamp of the version of the file to be opened,in **FILETIME** format as specified in [MS-DTYP] section 2.3.1. If no version of this file exists at this time stamp, the operation MUST be failed.

2.2.14 SMB2 CREATE Response

The SMB2 CREATE Response packet is sent by the server to notify the client of the status of its SMB2 CREATE Request. This response is composed of an SMB2 header, as specified in section 2.2.1, followed by this response structure:



...
...
...
CreateContextsOffset
CreateContextsLength
Buffer (variable)
...

StructureSize (2 bytes): The server MUST set this field to 89, indicating the size of the request structure, not including the header. The server MUST set this field to this value regardless of how long **Buffer**[] actually is in the request being sent.

OplockLevel (1 byte): The oplock level that is granted to the client for this open. This field MUST contain one of the following values:<20>

Value	Meaning
SMB2_OPLOCK_LEVEL_NONE 0x00	No oplock was granted.
SMB2_OPLOCK_LEVEL_II 0x01	A level II oplock was granted.
SMB2_OPLOCK_LEVEL_EXCLUSIVE 0x08	An exclusive oplock was granted.
SMB2_OPLOCK_LEVEL_BATCH 0x09	A batch oplock was granted.

Reserved (1 byte): Unused at the present and MUST be treated as reserved. The server sets this to any value, and the client MUST ignore it on receipt.

CreateAction (4 bytes): The action taken in establishing the open. This field MUST contain one of the following values:

Value	Meaning
FILE_SUPERSEDED 0x00000000	An existing file was deleted and a new file was created in its place.
FILE_OPENED 0x00000001	An existing file was opened.
FILE_CREATED 0x00000002	A new file was created.

Value	Meaning
FILE_OVERWRITTEN 0x00000003	An existing file was overwritten.

CreationTime (8 bytes): The time when the file was created; in **FILETIME** format as specified in [MS-DTYP] section 2.3.1.

LastAccessTime (8 bytes): The time the file was last accessed; in **FILETIME** format as specified in [MS-DTYP] section 2.3.1.

LastWriteTime (8 bytes): The time when data was last written to the file; in **FILETIME** format as specified in [MS-DTYP] section 2.3.1.

ChangeTime (8 bytes): The time when the file was last modified; in **FILETIME** format as specified in [MS-DTYP] section 2.3.1.

AllocationSize (8 bytes): The size, in bytes, of the data that is allocated to the file.

EndofFile (8 bytes): The size, in bytes, of the file.

FileAttributes (4 bytes): The attributes of the file. The valid flags are as specified in [MS-FSCC] section 2.6.

Reserved2 (4 bytes): Unused at the present and MUST be treated as reserved. The server sets this to any value, and the client MUST ignore it on receipt.

FileId (16 bytes): An SMB2_FILEID, as specified in section 2.2.14.1.

The identifier of the open to a file or pipe that was established.

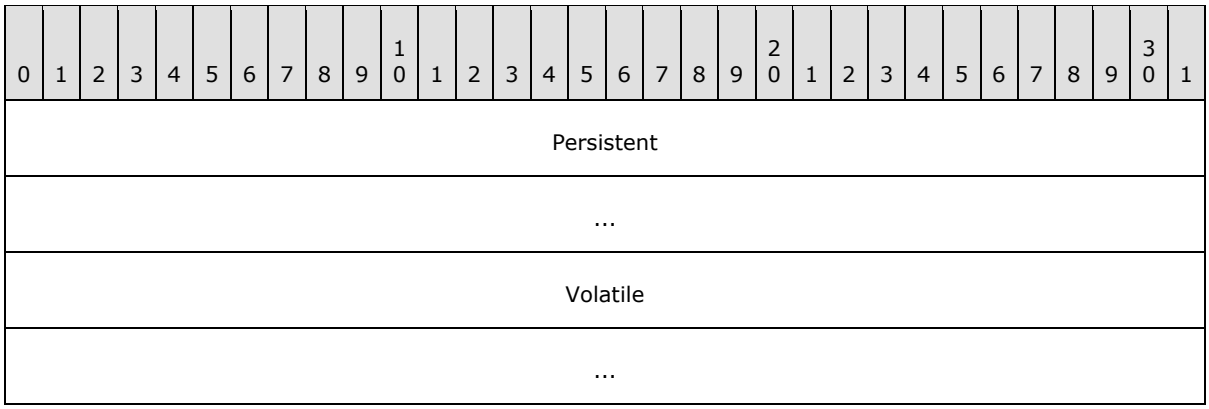
CreateContextsOffset (4 bytes): The offset, in bytes, from the beginning of the SMB2 header to the first 8-byte aligned SMB2_CREATE_CONTEXT response that is contained in this response. If none are being returned in the response, this value MUST be 0. These values are specified in section 2.2.14.2.

CreateContextsLength (4 bytes): The length, in bytes, of the list of SMB2_CREATE_CONTEXT response structures that are contained in this response.

Buffer (variable): A variable-length buffer that contains the list of create contexts that are contained in this response, as described by **CreateContextsOffset** and **CreateContextsLength**. This takes the form of a list of SMB2 CREATE_CONTEXT Response Values, as specified in section 2.2.14.2.

2.2.14.1 SMB2_FILEID

The SMB2 FILEID is used to represent an open to a file.



Persistent (8 bytes): A file handle that remains persistent when an open is reconnected after being lost on a disconnect, as specified in section 3.3.5.9.7. The server MUST return this file handle as part of an SMB2 CREATE Response (section 2.2.14).

Volatile (8 bytes): A file handle that MAY be changed when an open is reconnected after being lost on a disconnect, as specified in section 3.3.5.9.7. The server MUST return this file handle as part of an SMB2 CREATE Response (section 2.2.14). This value MUST NOT change unless a reconnection is performed.

2.2.14.2 SMB2_CREATE_CONTEXT Response Values

The SMB2 CREATE_CONTEXT Response Values MUST take the same form as specified in section 2.2.13.2. The individual values that are contained in the data buffer of the create context responses varies, based on the name of the create context in the request. For each well-known name that is specified in the definition of the NameOffset field in section 2.2.13, representing a type of SMB2_CREATE_CONTEXT request, the format of the response is provided below.

2.2.14.2.1 SMB2_CREATE_EA_BUFFER

The SMB_CREATE_EA_BUFFER request does not generate an SMB2 CREATE_CONTEXT Response.

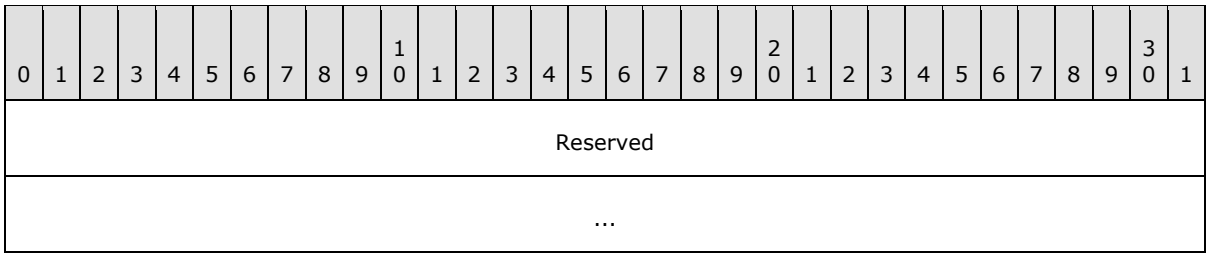
2.2.14.2.2 SMB2_CREATE_SD_BUFFER

The SMB2 CREATE_SD_BUFFER request does not generate an SMB2 CREATE_CONTEXT Response.

2.2.14.2.3 SMB2_CREATE_DURABLE_HANDLE_RESPONSE

If the server succeeds in opening a durable handle to a file as requested by the client via the SMB2_CREATE_DURABLE_HANDLE_REQUEST (section 2.2.13.2.3), it MUST send an SMB2 CREATE_DURABLE_HANDLE_RESPONSE back to the client to inform the client that the handle is durable.

If the server does not mark it for durable operation or the server does not implement durable handles, then it MUST ignore this request.



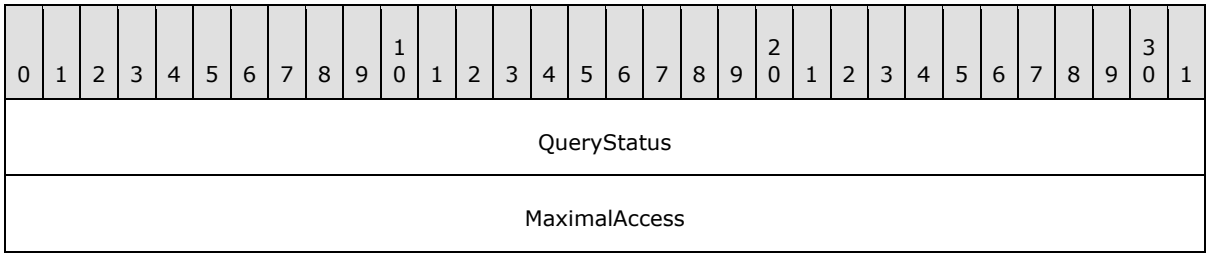
Reserved (8 bytes): Unused at the present and MUST be treated as reserved. The server MUST set this to 0, and the client MUST ignore it on receipt.

2.2.14.2.4 SMB2_CREATE_DURABLE_HANDLE_RECONNECT

The SMB2 CREATE_DURABLE_HANDLE_RECONNECT (section 2.2.13.2.4) request SHOULD NOT generate an SMB2 CREATE_CONTEXT Response.<21>

2.2.14.2.5 SMB2_CREATE_QUERY_MAXIMAL_ACCESS

The SMB2 CREATE_QUERY_MAXIMAL_ACCESS returns an SMB2_CREATE_CONTEXT in the response with the Name that is identified by SMB2_CREATE_QUERY_MAXIMAL ACCESS as specified in SMB2_CREATE_CONTEXT Request Values (section 2.2.13.2), if the server attempts to query maximal access as part of processing a create request.



QueryStatus (4 bytes): The resulting status code of the attempt to query maximal access. The **MaximalAccess** field is valid only if **QueryStatus** is STATUS_SUCCESS. The status code MUST be one of those defined in [MS-ERREF] section 2.3.

MaximalAccess (4 bytes): The maximal access that the user who is described by SessionId has on the file or named pipe that was opened. This is an access mask value, as specified in section 2.2.13.1.

2.2.14.2.6 SMB2_CREATE_ALLOCATION_SIZE

The SMB2 CREATE_ALLOCATION_SIZE request does not generate an SMB2 CREATE_CONTEXT Response.

2.2.14.2.7 SMB2_CREATE_TIMEWARP_TOKEN

The SMB2 CREATE_TIMEWARP_TOKEN request does not generate an SMB2 CREATE_CONTEXT Response.

2.2.14.2.8 SMB2_CREATE_QUERY_ON_DISK_ID

The server responds with an opaque 32-byte BLOB that uniquely identifies the opened file on disk. This value cannot change for the lifetime of the file.

2.2.15 SMB2 CLOSE Request

The SMB2 CLOSE Request packet is used by the client to close an instance of a file that was opened previously with a successful SMB2 CREATE Request. This request is composed of an SMB2 header, as specified in section 2.2.1, followed by this request structure:

StructureSize (2 bytes): The client MUST set this field to 24, indicating the size of the request structure, not including the header.

Flags (2 bytes): A Flags field, which indicates how the operation MUST be processed. This field MUST be constructed using the following values:

Value	Meaning
SMB2_CLOSE_FLAG_POSTQUERY_ATTRIB 0x0001	If set, the server MUST set the attribute fields in the response, as specified in section 2.2.16, to valid values. If not set, the client MUST NOT use the values that are returned in the response.

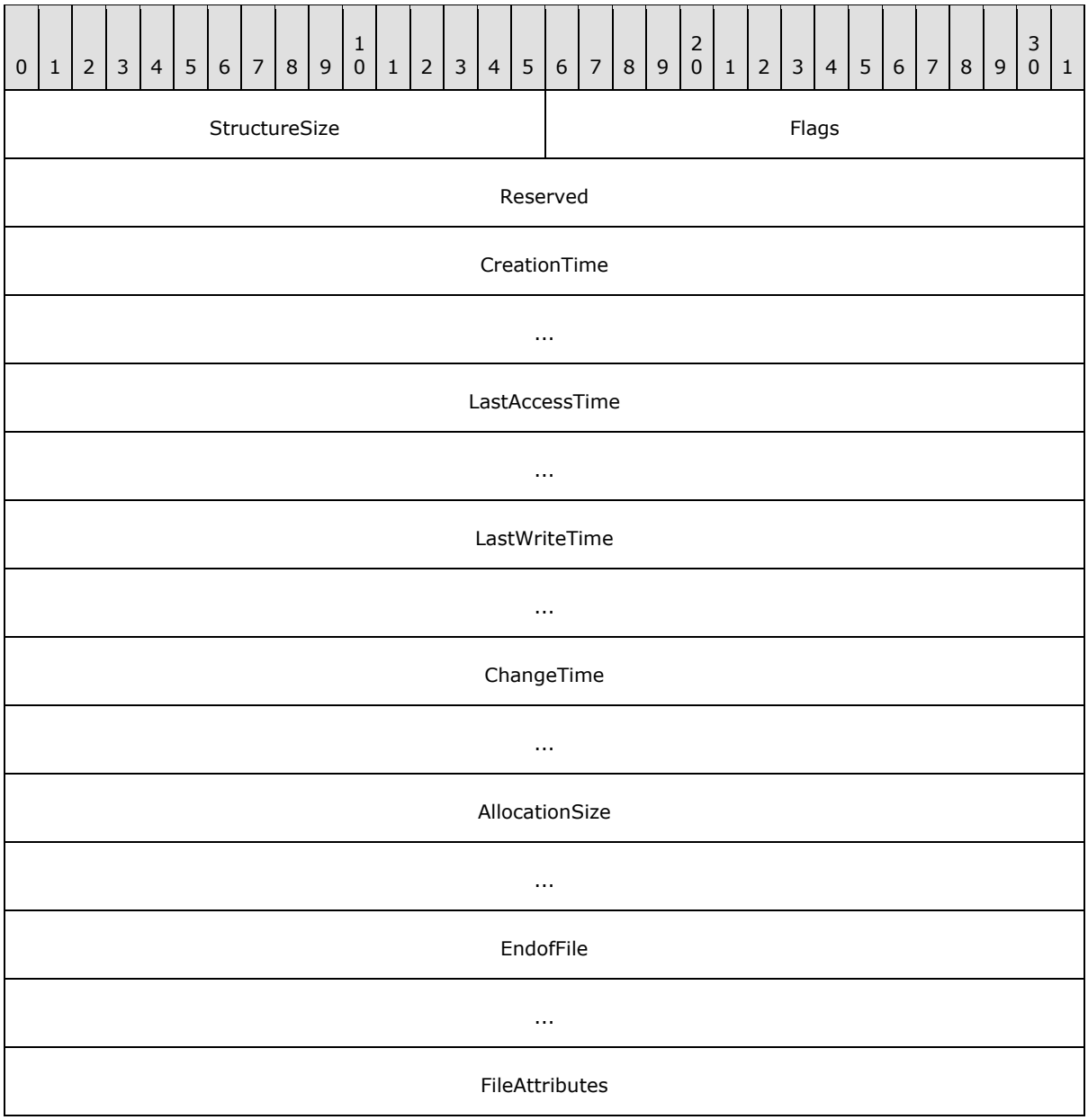
Reserved (4 bytes): Unused at the present, and MUST be treated as reserved. The client MUST set this to 0, and the server MUST ignore it on receipt.

FileId (16 bytes): An SMB2_FILEID structure, as specified in section 2.2.14.1.

The identifier of the open to a file or named pipe that is being closed.

2.2.16 SMB2 CLOSE Response

The SMB2 CLOSE Response packet is sent by the server to indicate that an SMB2 CLOSE Request was processed successfully. This response is composed of an SMB2 header, as specified in section 2.2.1, followed by this response structure:



StructureSize (2 bytes): The server MUST set this field to 60, indicating the size of the response structure, not including the header.

Flags (2 bytes): A Flags field that indicates how the operation MUST be processed. This field MUST be constructed using the following values:

Value	Meaning
SMB2_CLOSE_FLAG_POSTQUERY_ATTRIB 0x0001	If set, the client MUST use the attribute fields in the response. If not set, the client MUST NOT use the attribute fields that are returned in the response.

Reserved (4 bytes): Unused at present, and MUST be treated as reserved. The server MUST set this to 0, and the client MUST ignore it on receipt.

CreationTime (8 bytes): The time when the file was created; in **FILETIME** format as specified in [MS-DTYP] section 2.3.1. If the SMB2_CLOSE_FLAG_POSTQUERY_ATTRIB flag in the SMB2 CLOSE Request was set, this field MUST be set to the value that is returned by the attribute query. If the flag is not set, the field SHOULD be set to zero and MUST NOT be checked on receipt.

LastAccessTime (8 bytes): The time when the file was last accessed; in **FILETIME** format as specified in [MS-DTYP] section 2.3.1. If the SMB2_CLOSE_FLAG_POSTQUERY_ATTRIB flag in the SMB2 CLOSE Request was set, this field MUST be set to the value that is returned by the attribute query. If the flag is not set, this field MUST be set to zero.

LastWriteTime (8 bytes): The time when data was last written to the file; in **FILETIME** format as specified in [MS-DTYP] section 2.3.1. If the SMB2_CLOSE_FLAG_POSTQUERY_ATTRIB flag in the SMB2 CLOSE Request was set, this field MUST be set to the value that is returned by the attribute query. If the flag is not set, this field MUST be set to zero.

ChangeTime (8 bytes): The time when the file was last modified; in **FILETIME** format as specified in [MS-DTYP] section 2.3.1. If the SMB2_CLOSE_FLAG_POSTQUERY_ATTRIB flag in the SMB2 CLOSE Request was set, this field MUST be set to the value that is returned by the attribute query. If the flag is not set, this field MUST be set to zero.

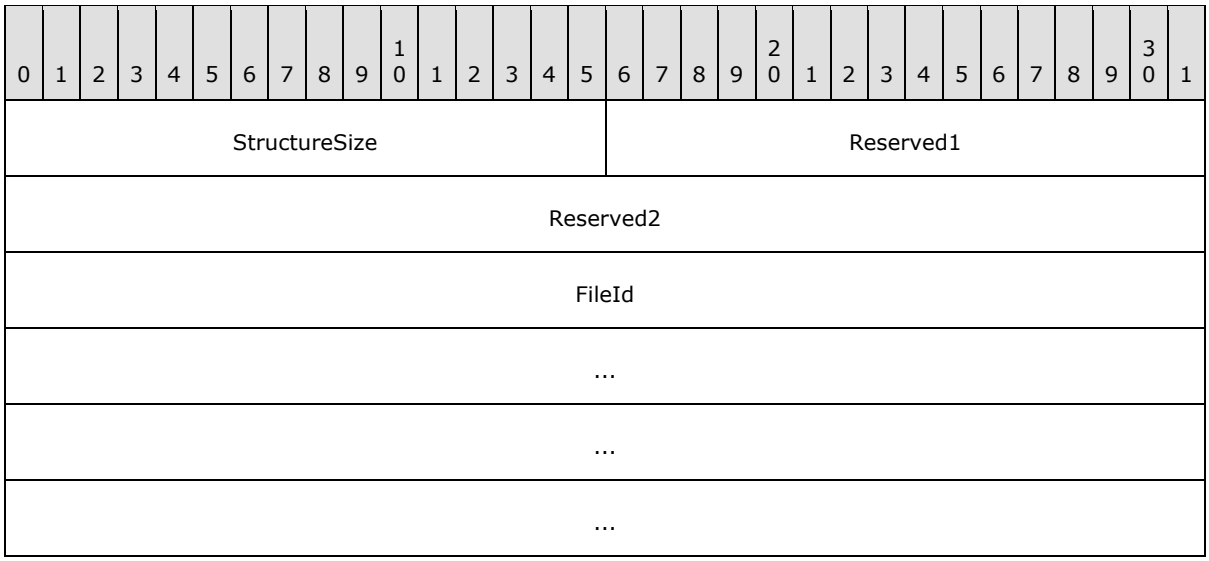
AllocationSize (8 bytes): The size, in bytes, of the data that is allocated to the file. If the SMB2_CLOSE_FLAG_POSTQUERY_ATTRIB flag in the SMB2 CLOSE Request was set, this field MUST be set to the value that is returned by the attribute query. If the flag is not set, this field MUST be set to zero.

EndofFile (8 bytes): The size, in bytes, of the file. If the SMB2_CLOSE_FLAG_POSTQUERY_ATTRIB flag in the SMB2 CLOSE Request was set, this field MUST be set to the value that is returned by the attribute query. If the flag is not set, this field MUST be set to zero.

FileAttributes (4 bytes): The attributes of the file. If the SMB2_CLOSE_FLAG_POSTQUERY_ATTRIB flag in the SMB2 CLOSE Request was set, this field MUST be set to the value that is returned by the attribute query. If the flag is not set, this field MUST be set to zero. For more information about valid flags, see [MS-FSCC] section 2.6.

2.2.17 SMB2 FLUSH Request

The SMB2 FLUSH Request packet is sent by a client to request that a server flush all cached file information for a specified open of a file to the persistent store that backs the file. If the open refers to a named pipe, the operation will complete once all data written to the pipe has been consumed by a reader. This request is composed of an SMB2 header, as specified in section 2.2.1, followed by this request structure:



StructureSize (2 bytes): The client MUST set this field to 30, indicating the size of the request structure, not including the header.

Reserved1 (2 bytes): Unused at the present and MUST be treated as reserved. The client MUST set this to 0, and the server MUST ignore it on receipt.

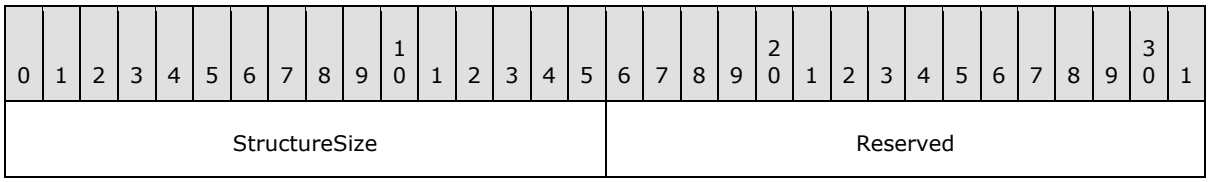
Reserved2 (4 bytes): Unused at the present and MUST be treated as reserved. The client MUST set this to 0, and the server MUST ignore it on receipt.

FileId (16 bytes): An SMB2_FILEID, as specified in section 2.2.14.1.

The client MUST set this field to the identifier of the open to a file or named pipe that is being flushed.

2.2.18 SMB2 FLUSH Response

The SMB2 FLUSH Response packet is sent by the server to confirm that an SMB2 FLUSH Request (section 2.2.17) was successfully processed. This response is composed of an SMB2 header, as specified in section 2.2.1, followed by this request structure:

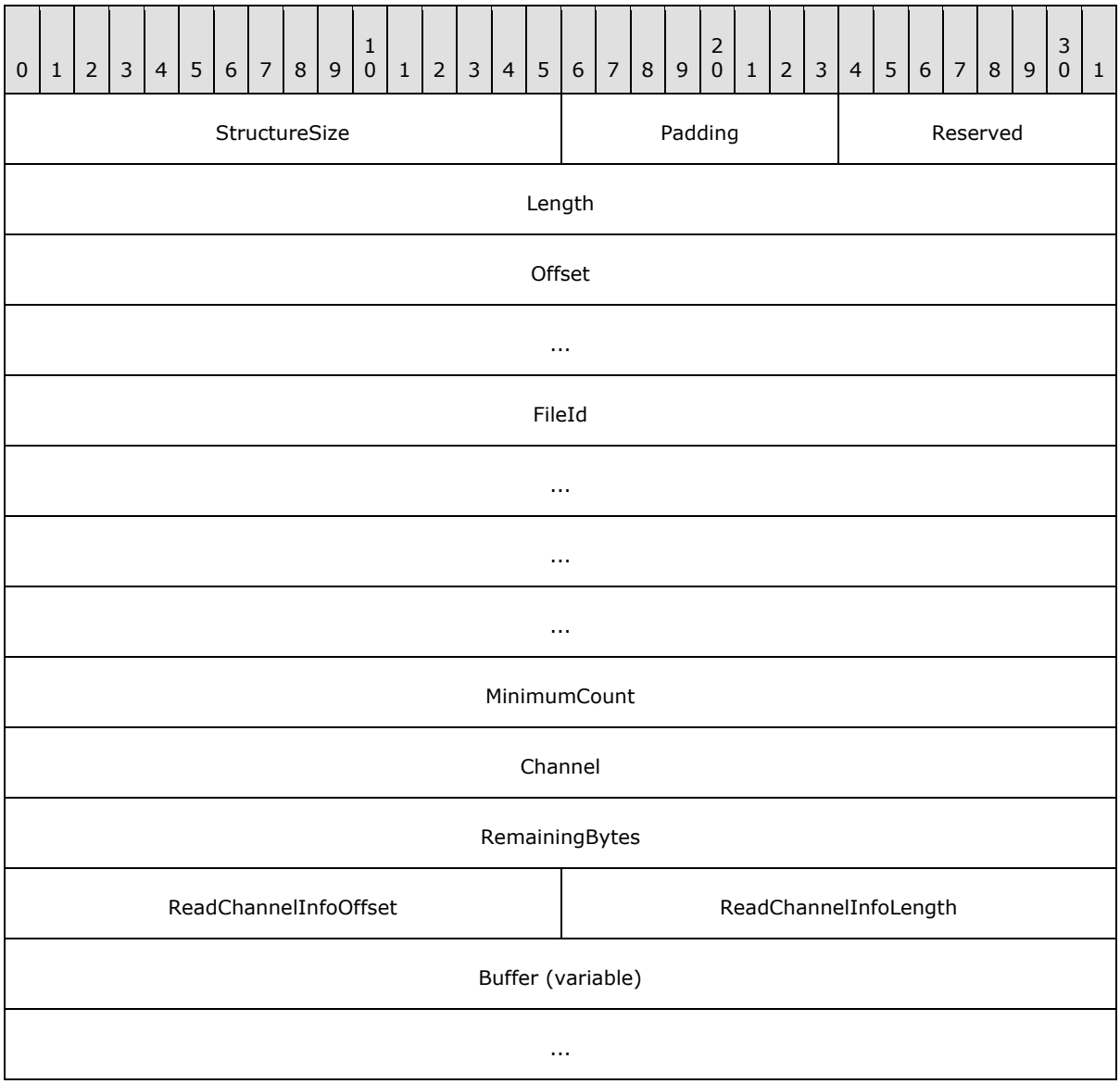


StructureSize (2 bytes): The server MUST set this field to 4, indicating the size of the response structure, not including the header.

Reserved (2 bytes): Unused at the present and MUST be treated as reserved. The server MUST set this field to 0, and the client MUST ignore it on receipt.

2.2.19 SMB2 READ Request

The SMB2 READ Request packet is sent by the client to request a read operation on the file that is specified by the **FileId**. This request is composed of an SMB2 header, as specified in section 2.2.1, followed by this request structure:



StructureSize (2 bytes): The client MUST set this field to 49, indicating the size of the request structure, not including the header. The client MUST set it to this value regardless of how long **Buffer**[] actually is in the request being sent.

Padding (1 byte): The requested offset, in bytes, to place the data read in the SMB2 READ Response (section 2.2.20). This value is provided to optimize data placement on the client and is not binding on the server.

Reserved (1 byte): Unused at the present and MUST be treated as reserved. The client MUST set this field to 0, and the server MUST ignore it on receipt.

Length (4 bytes): The length, in bytes, of the data to read from the specified file or pipe. The length of the data being read may be zero bytes.

Offset (8 bytes): The offset, in bytes, into the file from which the data MUST be read. If the read is being executed on a pipe, the Offset MUST be set to 0 by the client and MUST be ignored by the server.

FileId (16 bytes): An SMB2_FILEID, as specified in section 2.2.14.1.

The identifier of the file or pipe on which to perform the read.

MinimumCount (4 bytes): The minimum number of bytes to be read for this operation to be successful. If fewer than the minimum number of bytes are read by the server, the server MUST return an error rather than the bytes read.

Channel (4 bytes): Unused at the present and MUST be treated as reserved. The client MUST set this field to 0, and the server MUST ignore it on receipt.

RemainingBytes (4 bytes): The number of subsequent bytes that the client intends to read from the file after this operation completes. This value is provided to facilitate read-ahead caching, and is not binding on the server.

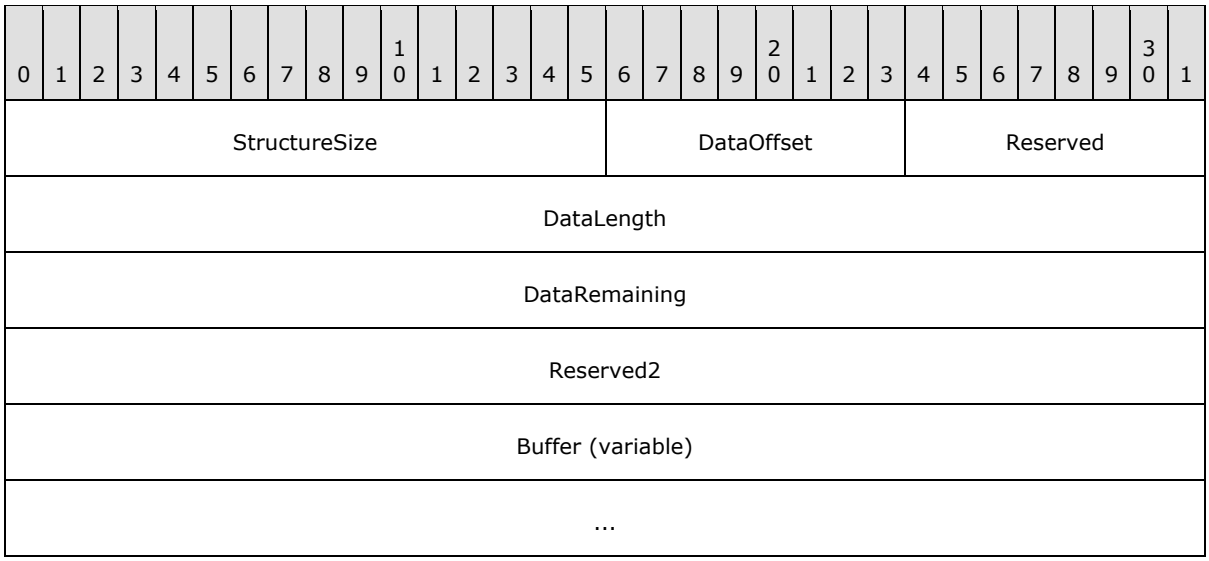
ReadChannelInfoOffset (2 bytes): Unused at the present and MUST be treated as reserved. The client MUST set this field to 0, and the server MUST ignore it on receipt.

ReadChannelInfoLength (2 bytes): Unused at the present and MUST be treated as reserved. The client MUST set this field to 0, and the server MUST ignore it on receipt.

Buffer (variable): A variable-length buffer that contains the read channel information, as described by **ReadChannelInfoOffset** and **ReadChannelInfoLength**. Unused at present. The client MUST set one byte of this field to 0, and the server MUST ignore it on receipt.

2.2.20 SMB2 READ Response

The SMB2 READ Response packet is sent in response to an SMB2 READ Request (section 2.2.19) packet. This response is composed of an SMB2 header, as specified in section 2.2.1, followed by this response structure:



StructureSize (2 bytes): The server MUST set this field to 17, indicating the size of the response structure, not including the header. This value MUST be used regardless of how large **Buffer**[] is in the actual response.

DataOffset (1 byte): The offset, in bytes, from the beginning of the header to the data read being returned in this response.

Reserved (1 byte): Unused at the present and MUST be treated as reserved. The server MUST set this to 0, and the client MUST ignore it on receipt.

DataLength (4 bytes): The length, in bytes, of the data read being returned in this response. The minimum size is 1 byte.

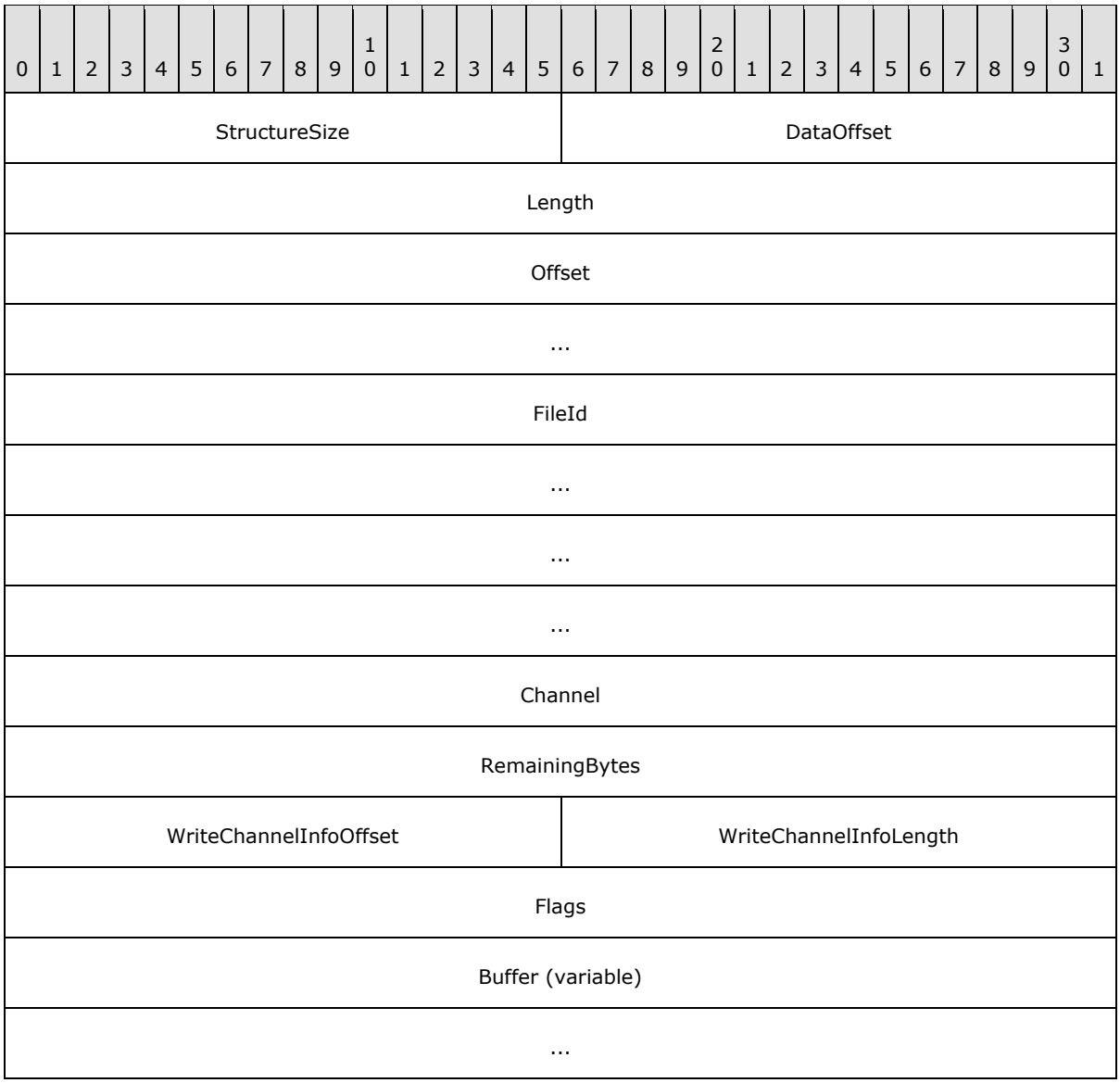
DataRemaining (4 bytes): Unused at the present and MUST be treated as reserved. The server MUST set this field to 0, and the client MUST ignore it on receipt.

Reserved2 (4 bytes): Unused at the present and MUST be treated as reserved. The server MUST set this to 0, and the client MUST ignore it on receipt.

Buffer (variable): A variable-length buffer that contains the data read for the response, as described by **DataOffset** and **DataLength**. The minimum length is 1 byte. If 0 bytes are returned from the file system, the server MUST send a failure response with status == STATUS_END_OF_FILE.

2.2.21 SMB2 WRITE Request

The SMB2 WRITE Request packet is sent by the client to write data to the file or named pipe on the server. This request is composed of an SMB2 header, as specified in section 2.2.1, followed by this request structure:



StructureSize (2 bytes): The server MUST set this field to 49, indicating the size of the request structure, not including the header. The server MUST set it to this value regardless of how long **Buffer**[] actually is in the request being sent.

DataOffset (2 bytes): The offset, in bytes, from the header where the data to be written is located in the request.

Length (4 bytes): The length of the data being written, in bytes. The length of the data being written may be zero bytes.

Offset (8 bytes): The offset, in bytes, of where to write the data in the destination file.

FileId (16 bytes): An SMB2_FILEID, as specified in section 2.2.14.1.

The identifier of the file or pipe on which to perform the write.

Channel (4 bytes): Unused at the present and MUST be treated as reserved. The client MUST set this field to 0, and the server MUST ignore it on receipt.

RemainingBytes (4 bytes): The number of subsequent bytes the client intends to write to the file after this operation completes. This value is provided to facilitate write caching and is not binding on the server.

WriteChannelInfoOffset (2 bytes): Unused at the present and MUST be treated as reserved. The client MUST set this field to 0, and the server MUST ignore it on receipt.

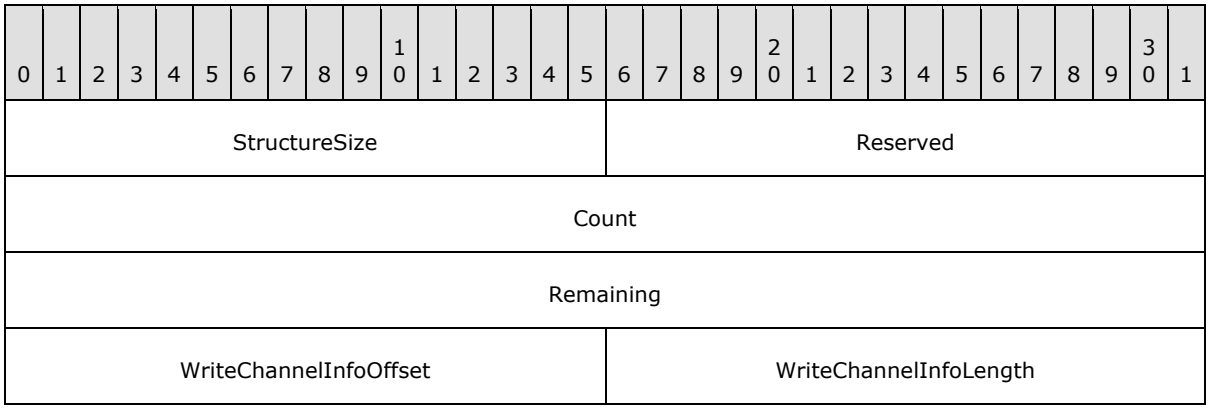
WriteChannelInfoLength (2 bytes): Unused at the present and MUST be treated as reserved. The client MUST set this field to 0, and the server MUST ignore it on receipt.

Flags (4 bytes): Unused at the present and MUST be treated as reserved. The client MUST set this field to 0, and the server MUST ignore it on receipt.

Buffer (variable): A variable-length buffer that contains the data to write and the write channel information, as described by **DataOffset**, **Length**, **WriteChannelInfoOffset**, and **WriteChannelInfoLength**.

2.2.22 SMB2 WRITE Response

The SMB2 WRITE Response packet is sent in response to an SMB2 WRITE Request (section 2.2.21) packet. This response is composed of an SMB2 header, as specified in section 2.2.1, followed by this response structure:



StructureSize (2 bytes): The server MUST set this field to 17, indicating the size of the response structure, not including the header.

Reserved (2 bytes): Unused at the present and MUST be treated as reserved. The server MUST set this to 0, and the client MUST ignore it on receipt.

Count (4 bytes): The number of bytes written.

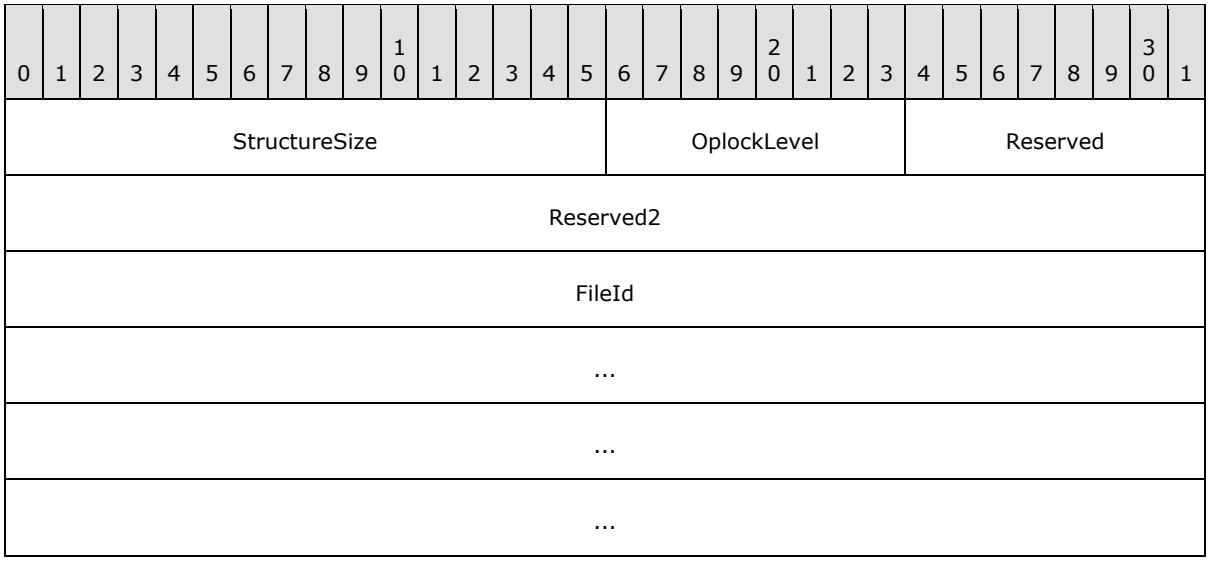
Remaining (4 bytes): Unused at the present and MUST be treated as reserved. The server MUST set this to 0, and the client MUST ignore it on receipt.

WriteChannelInfoOffset (2 bytes): Unused at the present and MUST be treated as reserved. The server MUST set this to 0, and the client MUST ignore it on receipt.

WriteChannelInfoLength (2 bytes): Unused at the present and MUST be treated as reserved. The server MUST set this to 0, and the client MUST ignore it on receipt.

2.2.23 SMB2 OPLOCK_BREAK Notification Packet

The SMB2 OPLOCK_BREAK Notification Packet is sent by the server when the underlying object store indicates that an opportunistic lock (oplock) is being broken, representing a change in the oplock level. This response is composed of an SMB2 header, as specified in section 2.2.1, followed by this notification structure:



StructureSize (2 bytes): The server MUST set this to 24, indicating the size of the response structure, not including the header.

OplockLevel (1 byte): The server MUST set this to the maximum value of the **OplockLevel** that the server will accept for an acknowledgment from the client. Because SMB2_OPLOCK_LEVEL_BATCH is the highest oplock level, and it is being broken to a lower level, the server will never send a break from SMB2_OPLOCK_LEVEL_BATCH to SMB2_OPLOCK_LEVEL_BATCH. Thus this field MUST contain one of the following values:<22>

Value	Meaning
SMB2_OPLOCK_LEVEL_NONE 0x00	No oplock is available.
SMB2_OPLOCK_LEVEL_II 0x01	A level II oplock is available.

Reserved (1 byte): Unused at the present and MUST be treated as reserved. The server MUST set this to 0, and the client MUST ignore it on receipt.

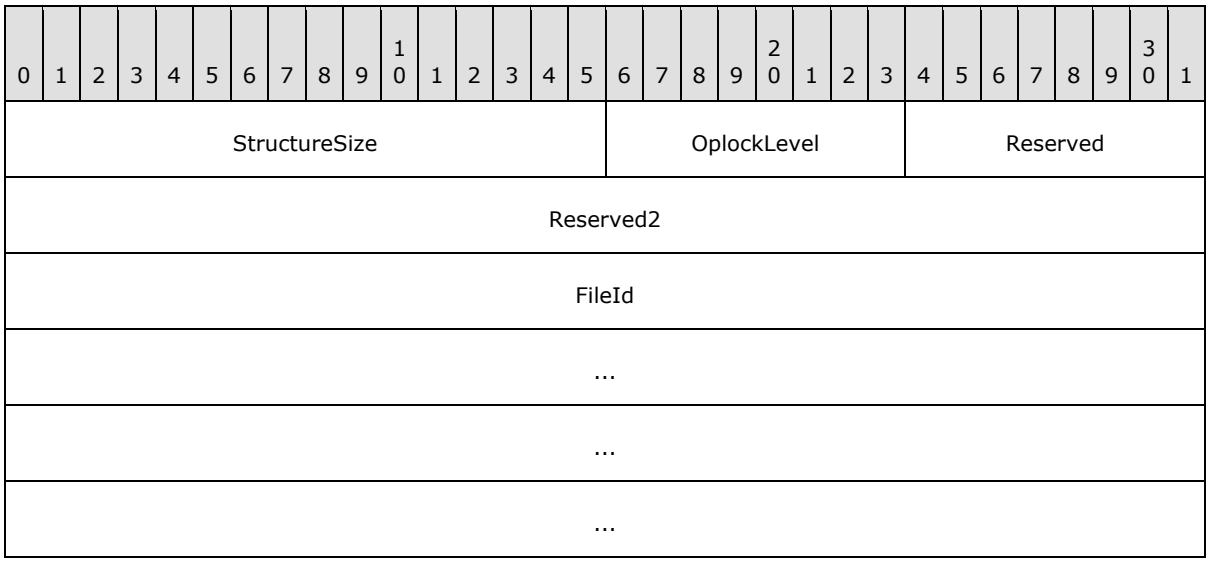
Reserved2 (4 bytes): Unused at the present and MUST be treated as reserved. The server MUST set this to 0, and the client MUST ignore it on receipt.

FileId (16 bytes): An SMB2_FILEID, as specified in section 2.2.14.1.

The identifier of the file or pipe which is either having its oplock broken, or its **oplock break** acknowledged.

2.2.24 SMB2 OPLOCK_BREAK Acknowledgment

The SMB2 OPLOCK_BREAK Acknowledgment packet is sent by the client in response to an SMB2 OPLOCK_BREAK notification packet sent by the server. The server responds to an oplock break acknowledgment with an SMB2 OPLOCK_BREAK response. The client MUST NOT send an oplock break acknowledgment for an oplock break from level II to none. A break from level II MUST always transition to none. Thus, the client does not have to send a request to the server because there is no question how the transition was made.



StructureSize (2 bytes): The client MUST set this to 24, indicating the size of the request structure, not including the header.

OplockLevel (1 byte): The resulting oplock level. This MUST be at least as permissive as the level that is specified by the server in its initial oplock break notification packet. For example, if the server specifies an SMB2_OPLOCK_LEVEL_II, the client can respond with an SMB2_OPLOCK_LEVEL_II or an SMB2_OPLOCK_LEVEL_NONE. Because SMB2_OPLOCK_LEVEL_BATCH is the highest oplock level, the server will never send a break from SMB2_OPLOCK_LEVEL_BATCH to SMB2_OPLOCK_LEVEL_BATCH. Thus this field MUST contain one of the following values:<23>

Value	Meaning
SMB2_OPLOCK_LEVEL_NONE 0x00	The client has lowered its oplock level for this file to none.
SMB2_OPLOCK_LEVEL_II 0x01	The client has lowered its oplock level for this file to level II.

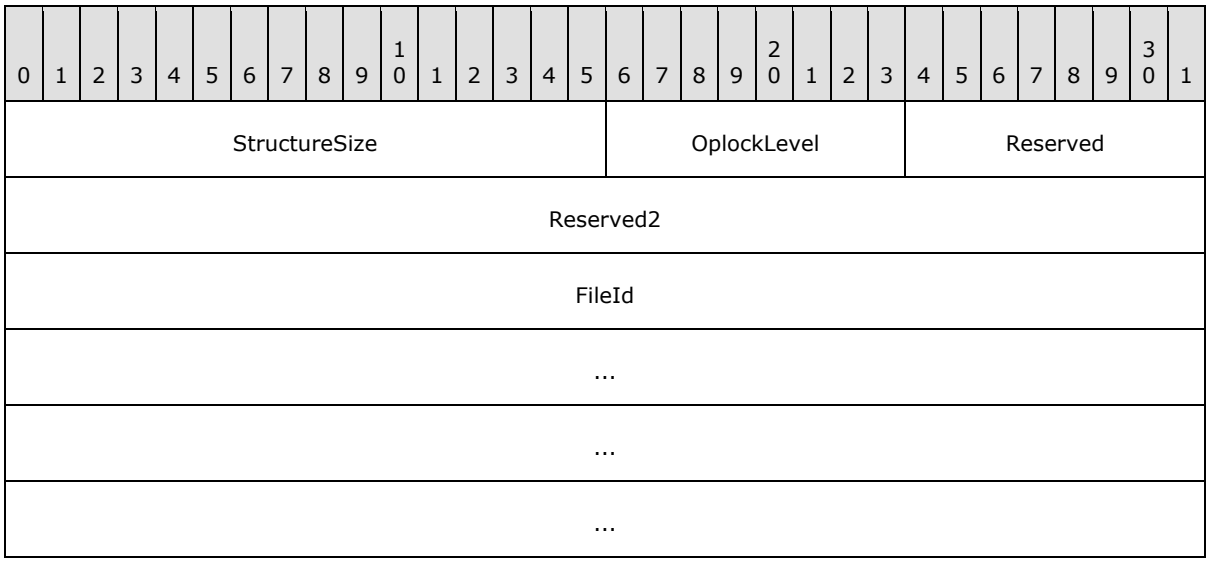
Reserved (1 byte): Unused at the present, and MUST be treated as reserved. The client MUST set this to 0, and the server MUST ignore it on receipt.

Reserved2 (4 bytes): Unused at the present, and MUST be treated as reserved. The client MUST set this to 0, and the server MUST ignore it on receipt.

FileId (16 bytes): An SMB2_FILEID that identifies the file or pipe on which the oplock break is being acknowledged.

2.2.25 SMB2 OPLOCK_BREAK Response

The SMB2 OPLOCK_BREAK Response packet is sent by the server in response to an oplock break acknowledgment from the client. This response is composed of an SMB2 header, as specified in section 2.2.1, followed by this response structure:



StructureSize (2 bytes): The server MUST set this to 24, indicating the size of the response structure, not including the header.

OplockLevel (1 byte): The server MUST set this value to the OplockLevel value in the SMB2 OPLOCK BREAK (section 2.2.24) acknowledgment that is sent by the client as specified in section 2.2.24. Because SMB2_OPLOCK_LEVEL_BATCH is the highest oplock level, the server will never send a break from SMB2_OPLOCK_LEVEL_BATCH to SMB2_OPLOCK_LEVEL_BATCH. Thus this field MUST contain one of the following values:<24>

Value	Meaning
SMB2_OPLOCK_LEVEL_NONE 0x00	No oplock is available.
SMB2_OPLOCK_LEVEL_II 0x01	A level II oplock is available.

Reserved (1 byte): Unused at the present and MUST be treated as reserved. The server MUST set this to 0, and the client MUST ignore it on receipt.

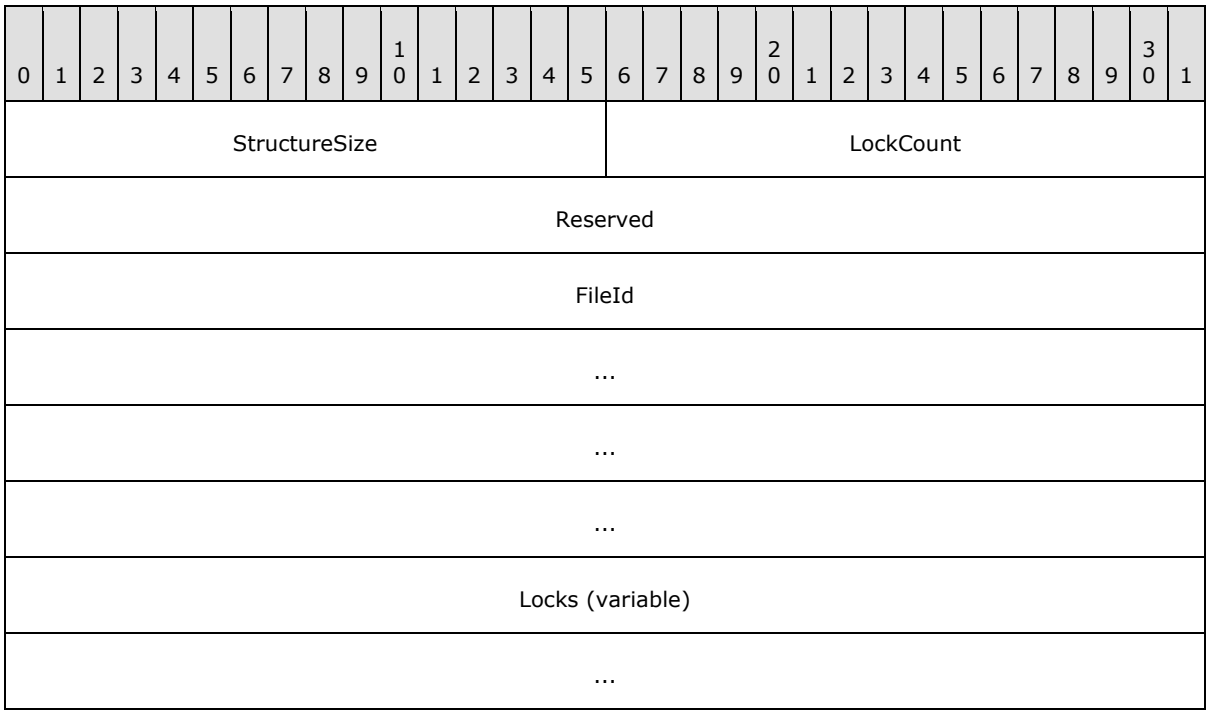
Reserved2 (4 bytes): Unused at the present and MUST be treated as reserved. The server MUST set this to 0, and the client MUST ignore it on receipt.

FileId (16 bytes): An SMB2_FILEID, as specified in section 2.2.14.1.

The identifier of the file or pipe, which MUST be the same as the FileId in the client's oplock break acknowledgment packet as specified in section 2.2.24.

2.2.26 SMB2 LOCK Request

The SMB2 LOCK Request packet is sent by the client to either lock or unlock portions of a file. Several different segments of the file can be affected with a single SMB2 LOCK Request packet, but they all must be within the same file.



StructureSize (2 bytes): The client MUST set this to 48, indicating the size of an SMB2_LOCK response with a single SMB2_LOCK structure. This value is set regardless of the number of locks that are sent.

LockCount (2 bytes): MUST be set to the number of SMB2_LOCK structures that are contained in the Locks[] array. The lock count MUST be greater than or equal to 1.

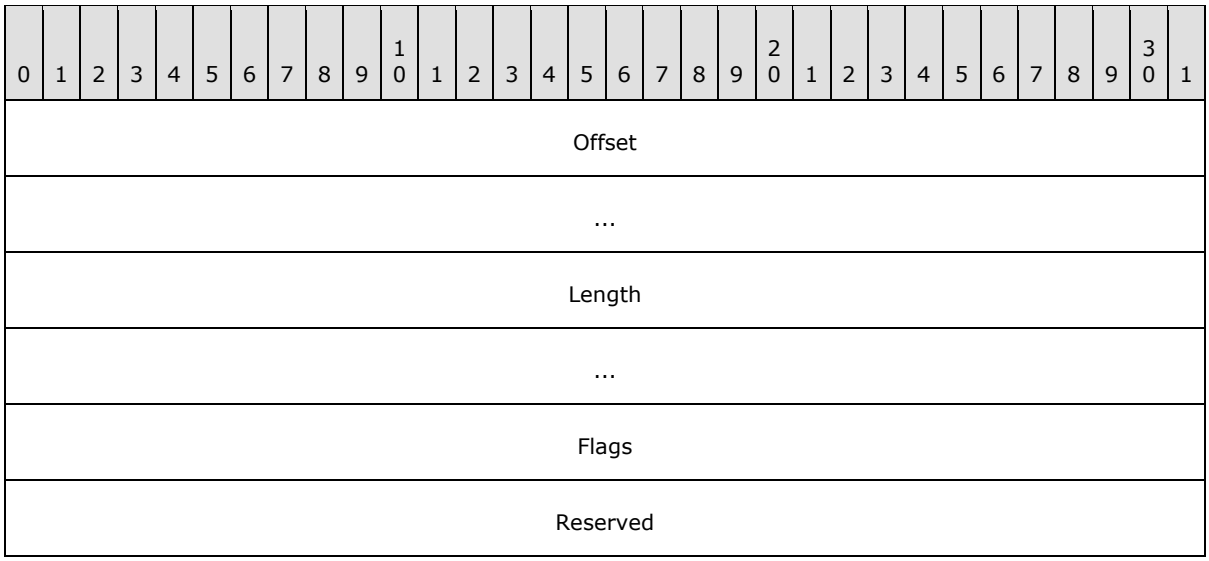
Reserved (4 bytes): Unused at the present, and MUST be treated as reserved. The client MUST set this to 0, and the server MUST ignore it on receipt.

FileId (16 bytes): An SMB2_FILEID that identifies the file on which to perform the byte range locks or unlocks.

Locks (variable): An array of **LockCount** (**SMB2_LOCK_ELEMENT**) structures that define the ranges to be locked or unlocked.

2.2.26.1 SMB2_LOCK_ELEMENT Structure

The SMB2 LOCK_ELEMENT Structure packet is used by the SMB2 LOCK Request packet to indicate segments of files that should be locked or unlocked.



Offset (8 bytes): The starting offset, in bytes, in the destination file from where the range being locked or unlocked starts.

Length (8 bytes): The length, in bytes, of the range being locked or unlocked.

Flags (4 bytes): The description of how the range is being locked or unlocked and how the operation must behave. This field takes the following format:

Value	Meaning
SMB2_LOCKFLAG_SHARED_LOCK 0x00000001	The range MUST be locked shared, allowing other opens to read from or take a shared lock on the range. Other opens MUST NOT be allowed to write within the range. Other locks can be requested and taken on this range.
SMB2_LOCKFLAG_EXCLUSIVE_LOCK 0x00000002	The range MUST be locked exclusive, not allowing other opens to read, write, or lock within the range.
SMB2_LOCKFLAG_UNLOCK 0x00000004	The range MUST be unlocked from a previous lock taken on this range. The unlock range MUST be identical to the lock range. Sub-ranges cannot be unlocked.
SMB2_LOCKFLAG_FAIL_IMMEDIATELY 0x00000010	The lock operation MUST fail immediately if it conflicts with an existing lock, instead of waiting for the range to become available.

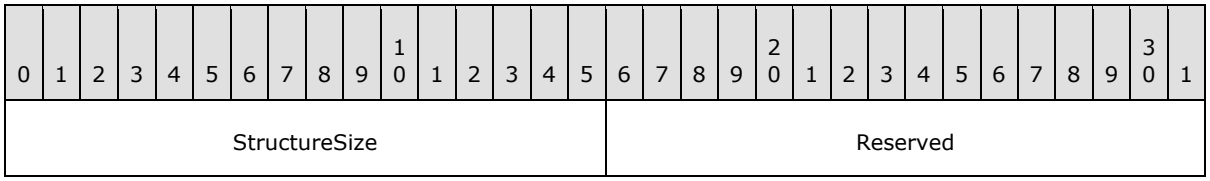
The following are the only valid combinations for the flags field:

- SMB2_LOCKFLAG_SHARED_LOCK
- SMB2_LOCKFLAG_EXCLUSIVE_LOCK
- SMB2_LOCKFLAG_SHARED_LOCK | SMB2_LOCKFLAG_FAIL_IMMEDIATELY
- SMB2_LOCKFLAG_EXCLUSIVE_LOCK | SMB2_LOCKFLAG_FAIL_IMMEDIATELY
- SMB2_LOCKFLAG_UNLOCK

Reserved (4 bytes): Unused at the present, and MUST be treated as reserved. The client MUST set this to 0, and the server MUST ignore it on receipt.

2.2.27 SMB2 LOCK Response

The SMB2 LOCK Response packet is sent by a server in response to an SMB2 LOCK Request (section 2.2.26) packet. This response is composed of an SMB2 header, as specified in section 2.2.1, followed by this request structure:

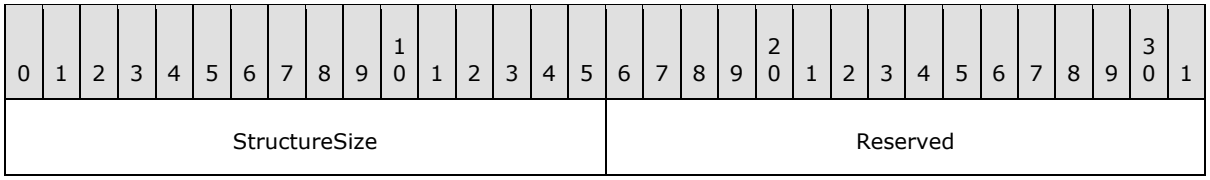


StructureSize (2 bytes): The server MUST set this to 4, indicating the size of the response structure, not including the header.

Reserved (2 bytes): Unused at the present and MUST be treated as reserved. The server MUST set this to 0, and the client MUST ignore it on receipt.

2.2.28 SMB2 ECHO Request

The SMB2 ECHO Request packet is sent by a client to determine whether a server is processing requests. This request is composed of an SMB2 header, as specified in section 2.2.1, followed by this request structure:

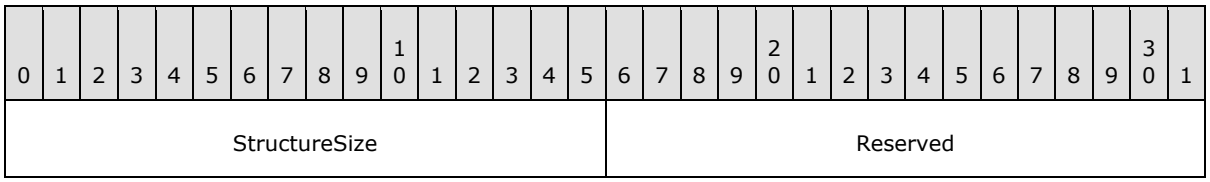


StructureSize (2 bytes): The client MUST set this to 4, indicating the size of the request structure, not including the header.

Reserved (2 bytes): Unused at the present and MUST be treated as reserved. The client MUST set this to 0, and the server MUST ignore it on receipt.

2.2.29 SMB2 ECHO Response

The SMB2 ECHO Response packet.

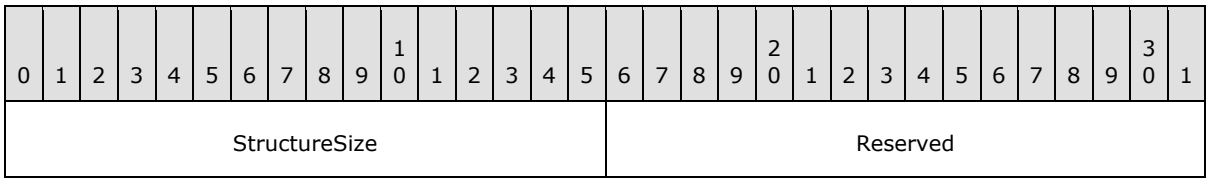


StructureSize (2 bytes): The server MUST set this to 4, indicating the size of the response structure, not including the header.

Reserved (2 bytes): Unused at the present and MUST be treated as reserved. The server MUST set this to 0, and the client MUST ignore it on receipt.

2.2.30 SMB2 CANCEL Request

The SMB2 CANCEL Request packet is sent by the client to cancel a previously sent message on the same SMB2 transport connection. The **MessageId** of the request to be canceled MUST be set in the SMB2 header of the request. This request is composed of an SMB2 header, as specified in section 2.2.1, followed by this request structure:

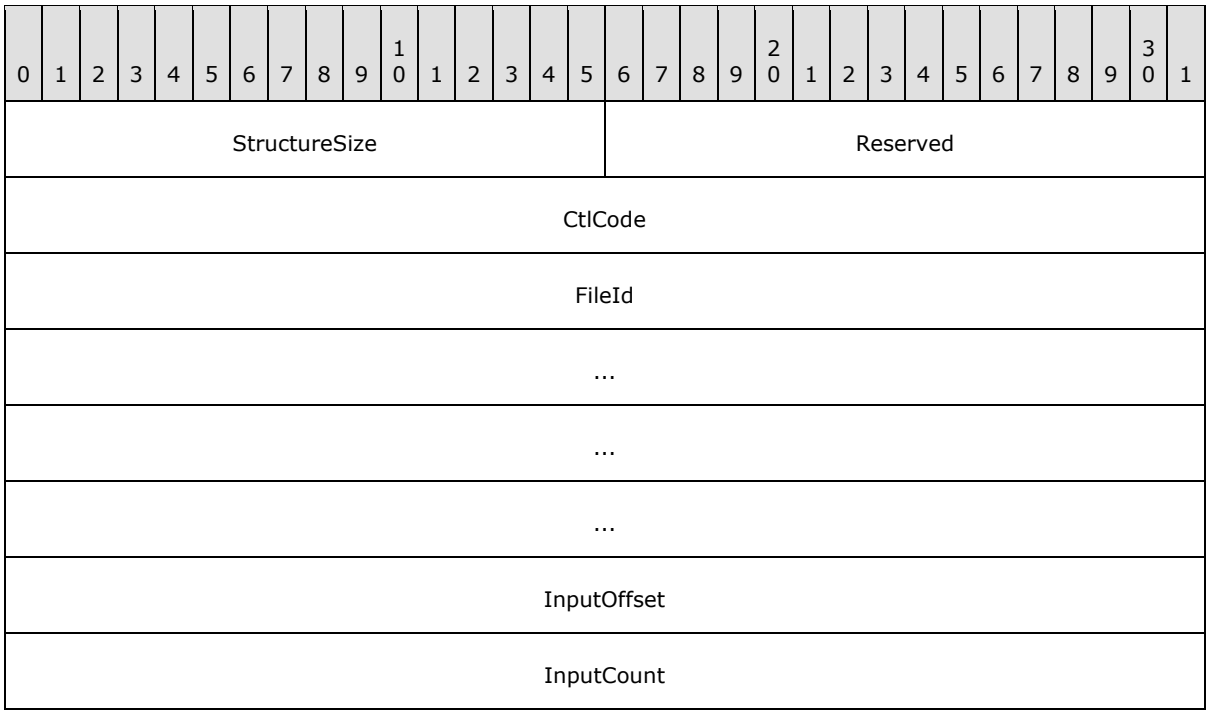


StructureSize (2 bytes): The client MUST set this field to 4, indicating the size of the request structure, not including the header.

Reserved (2 bytes): Unused at present and MUST be treated as reserved. The client MUST set this field to 0, and the server MUST ignore it on receipt.

2.2.31 SMB2 IOCTL Request

The SMB2 IOCTL Request packet is sent by a client to issue an implementation-specific **file system control** or device control (**FSCTL**/**IOCTL**) command across the network. For a list of IOCTL operations, see section 3.2.4.19 and [MS-FSCC] section 2.3. This request is composed of an SMB2 header, as specified in section 2.2.1, followed by this request structure:



MaxInputResponse
OutputOffset
OutputCount
MaxOutputResponse
Flags
Reserved2
Buffer (variable)
...

StructureSize (2 bytes): The client MUST set this field to 57, indicating the size of the request structure, not including the header. The client MUST set this field to this value regardless of how long **Buffer**[] actually is in the request being sent.

Reserved (2 bytes): Unused at present and MUST be treated as reserved. The client MUST set this field to 0, and the server MUST ignore it on receipt.

CtlCode (4 bytes): The control code of the FSCTL/IOCTL method. The values are listed in subsequent sections, and in [MS-FSCC] section 2.3. The following values indicate SMB2-specific processing as specified in sections 3.2.4.19 and 3.3.5.15:

Name	Value
FSCTL_DFS_GET_REFERRALS	0x00060194
FSCTL_PIPE_PEEK	0x0011400C
FSCTL_PIPE_TRANSCEIVE	0x0011C017
FSCTL_SRV_COPYCHUNK	0x001440F2
FSCTL_SRV_ENUMERATE_SNAPSHOTS	0x00144064
FSCTL_SRV_REQUEST_RESUME_KEY	0x00140078

FSCTL_PIPE_TRANSCEIVE is valid only on a named pipe with mode set to FILE_PIPE_MESSAGE_MODE as specified in [MS-FSCC] section 2.4.28

FileId (16 bytes): An SMB2_FILEID identifier of the file on which to perform the command. For certain IOCTL or FSCTL values, as specified in section 3.3.5.15, this parameter MUST be ignored.

InputOffset (4 bytes): The offset, in bytes, from the beginning of the SMB2 header to the input data buffer. If no input data is required for the FSCTL/IOCTL command being issued, the client SHOULD set this value to 0 and the server MUST ignore it on receipt. <25>

InputCount (4 bytes): The size, in bytes, of the input data.

MaxInputResponse (4 bytes): The maximum number of bytes that the server can return for the input data in the SMB2IOCTL response.

OutputOffset (4 bytes): Unused at the present, and MUST be treated as reserved. The sender SHOULD set this to 0, and the receiver MUST ignore it.<26>

OutputCount (4 bytes): Unused at the present, and MUST be treated as reserved. The sender MUST set this to 0, and the receiver MUST ignore it.

MaxOutputResponse (4 bytes): The maximum number of bytes that the server can return for the output data in the SMB2IOCTL response.

Flags (4 bytes): Flags indicating how the operation must be processed. This field MUST be constructed using the following values:

Value	Meaning
SMB2_0_IOCTL_IS_FSCTL 0x00000001	If set, the request is an FSCTL request. If not set, the request is an IOCTL request.

Reserved2 (4 bytes): Unused at present and MUST be treated as reserved. The client MUST set this field to 0, and the server MUST ignore it on receipt.

Buffer (variable): A variable-length buffer that contains the input and output data buffer for the request, as described by the **InputOffset**, **InputCount**, **OutputOffset**, and **OutputCount**. There is no minimum size restriction for this field as there can be FSCTLs with no input or output buffers. For FSCTL_SRV_COPYCHUNK, the format of this buffer is specified in SRV_COPYCHUNK_COPY. The Buffer format for FSCTL_DFS_GET_REFERRALS is specified in [MS-DFSC] section 2.2.2. The format of this buffer for all other FSCTLs is specified in the reference topic for the FSCTL being called.

2.2.31.1 SRV_COPYCHUNK_COPY

The SRV_COPYCHUNK_COPY packet is sent in an SMB2 IOCTL Request by the client to initiate a server-side copy of data. It is set as the contents of the input data buffer. This packet consists of 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
SourceKey																															
...																															
...																															
...																															
...																															
...																															
ChunkCount																															
Reserved																															
Chunks (variable)																															
...																															

SourceKey (24 bytes): A key, obtained from the server in a SRV_REQUEST_RESUME_KEY Response (section 2.2.32.3), that represents the source file for the copy.

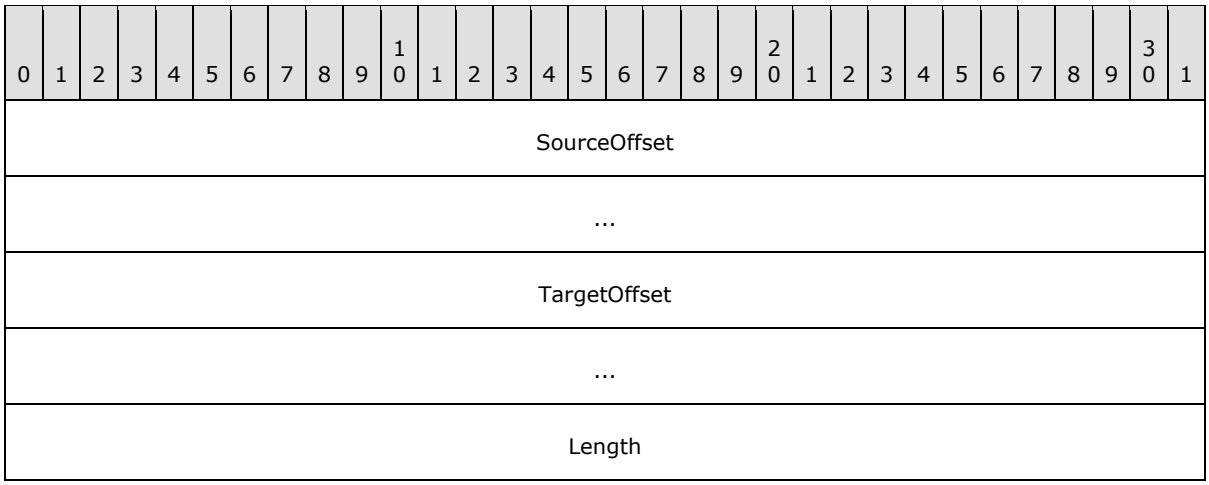
ChunkCount (4 bytes): The number of chunks of data that are to be copied.

Reserved (4 bytes): This field MUST be set to 0 by the client, and ignored by the server.

Chunks (variable): An array of packets describing the ranges to be copied. This array MUST be of a length equal to ChunkCount * size of SRV_COPYCHUNK.

2.2.31.1.1 SRV_COPYCHUNK

The SRV_COPYCHUNK packet is sent in the **Chunks** array of a SRV_COPYCHUNK_COPY packet to describe an individual data range to copy. This packet consists of the following:



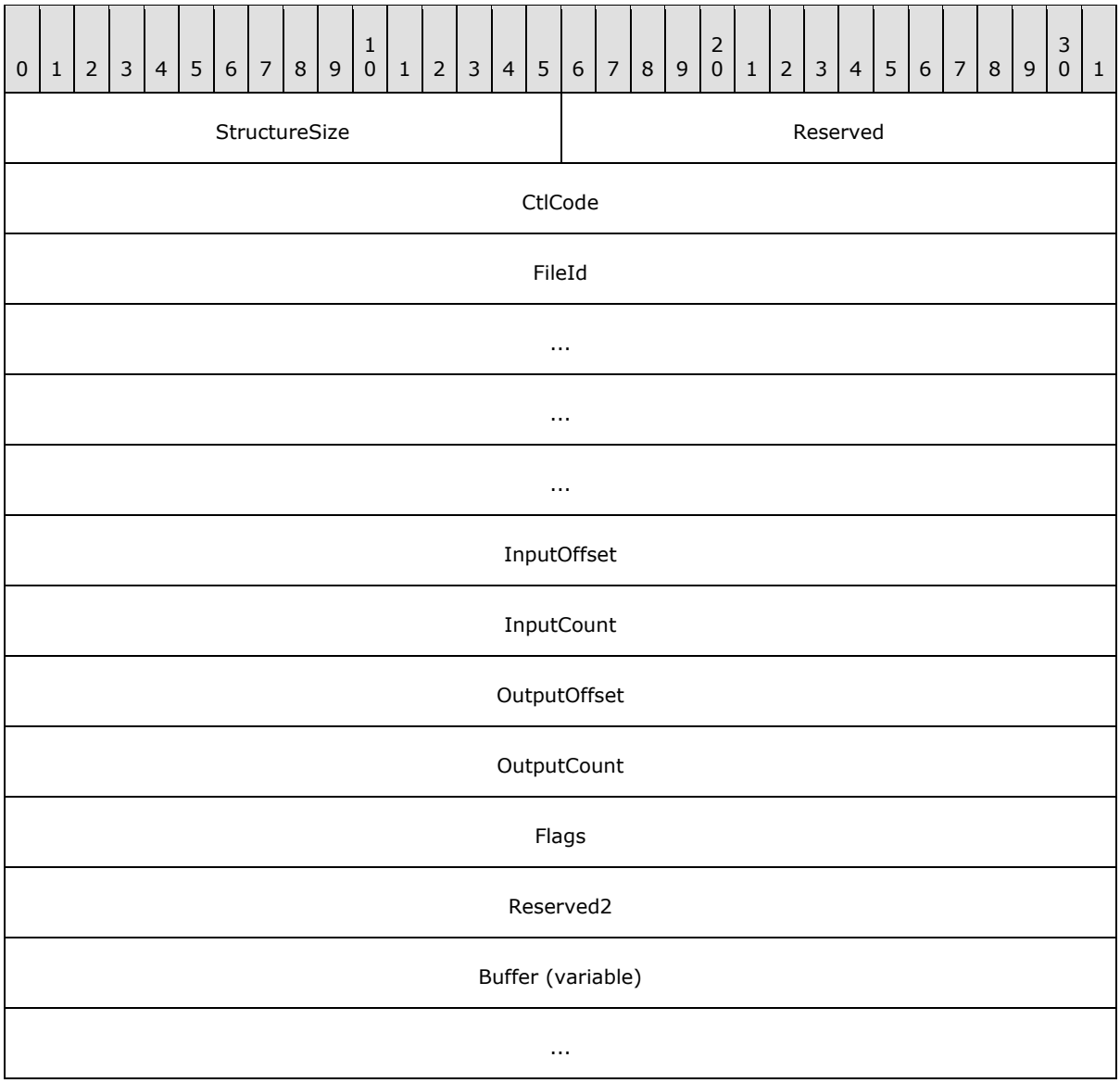
SourceOffset (8 bytes): The offset, in bytes, from the beginning of the source file to the location from which the data will be copied.

TargetOffset (8 bytes): The offset, in bytes, from the beginning of the destination file to where the data will be copied.

Length (4 bytes): The number of bytes of data to copy.

2.2.32 SMB2 IOCTL Response

The SMB2 IOCTL Response packet is sent by the server to transmit the results of a client SMB2 IOCTL request. This response consists of an SMB2 header, as specified in section 2.2.1, followed by this response structure:



StructureSize (2 bytes): The server MUST set this field to 49, indicating the size of the response structure, not including the header. This value MUST be used regardless of how large **Buffer**[] is in the actual response.

Reserved (2 bytes): Unused at present and MUST be treated as reserved. The server MUST set this field to 0, and the client MUST ignore it on receipt.

CtlCode (4 bytes): The control code of the FSCTL/IOCTL method that was executed.

FileId (16 bytes): An SMB2_FILEID identifier of the file on which the command was performed. For certain IOCTL or FSCTL values as specified in section 3.3.5.15, this parameter MUST be ignored.

InputOffset (4 bytes): Unused at the present, and MUST be treated as reserved. The sender SHOULD set this to 0, and the receiver MUST ignore it. <27>

InputCount (4 bytes): Unused at the present, and MUST be treated as reserved. The sender MUST set this to 0, and the receiver MUST ignore it.

OutputOffset (4 bytes): The offset, in bytes, from the beginning of the SMB2 header to the output data buffer. If no output data is returned for the FSCTL/IOCTL command that was issued, then this value MUST be set to 0.

OutputCount (4 bytes): The size, in bytes, of the output data.

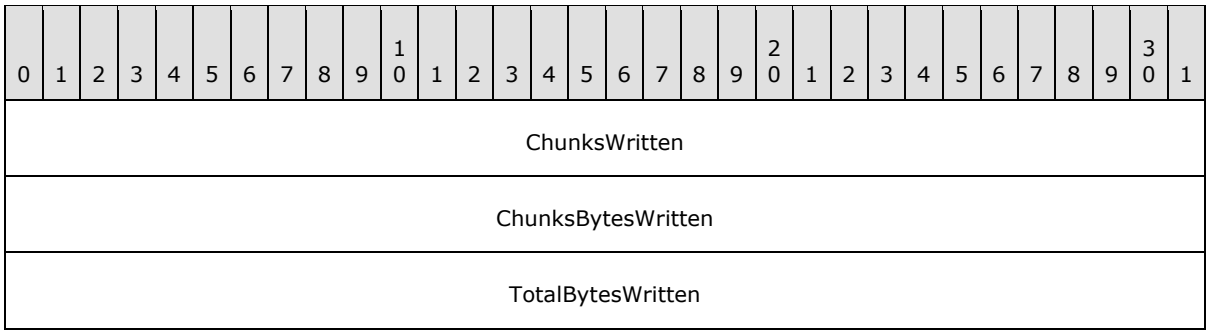
Flags (4 bytes): Unused at present and MUST be treated as reserved. The server MUST set this field to 0, and the client MUST ignore it on receipt.

Reserved2 (4 bytes): Unused at present and MUST be treated as reserved. The server MUST set this field to 0, and the client MUST ignore it on receipt.

Buffer (variable): A variable-length buffer that contains the input and output data buffer for the response, as described by **InputOffset**, **InputCount**, **OutputOffset**, and **OutputCount**. For more details, refer to section 3.3.5.15.

2.2.32.1 SRV_COPYCHUNK_RESPONSE

The SRV_COPYCHUNK_RESPONSE packet is sent in an SMB2 IOCTL Response by the server to return the results of a server-side copy operation. It is placed in the **Buffer** field of the SMB2 IOCTL Response packet. This packet consists of the following:



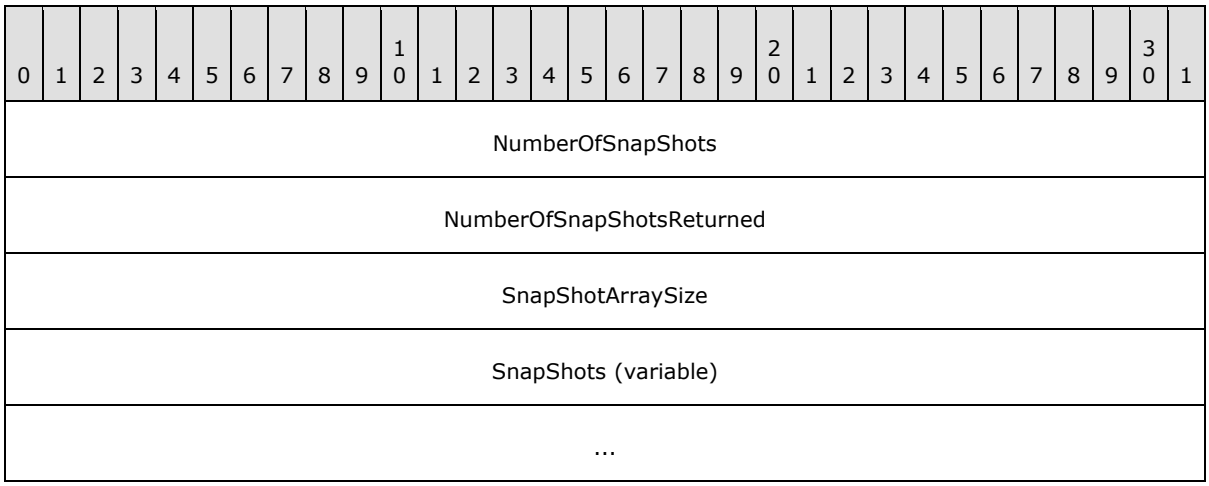
ChunksWritten (4 bytes): If the **Status** field in the SMB2 header of the response is not STATUS_INVALID_PARAMETER, as specified in [MS-ERREF] section 2.3, this value MUST indicate the number of chunks that were successfully written. If the **Status** field in the SMB2 header of the response is STATUS_INVALID_PARAMETER, this value MUST indicate the maximum number of chunks that the server will accept in a single request. This would allow the client to know how to correctly reissue the request.

ChunksBytesWritten (4 bytes): If the **Status** field in the SMB2 header of the response is not STATUS_INVALID_PARAMETER, as specified in [MS-ERREF] section 2.3, this value MUST indicate the number of bytes written in the last chunk that did not successfully process (if a partial write occurred). If the **Status** field in the SMB2 header of the response is STATUS_INVALID_PARAMETER, this value MUST indicate the maximum number of bytes the server will allow to be written in a single chunk.

TotalBytesWritten (4 bytes): If the **Status** field in the SMB2 header of the response is not STATUS_INVALID_PARAMETER, as specified in [MS-ERREF] section 2.3, this value MUST indicate the total number of bytes written in the server-side copy operation. If the **Status** field in the SMB2 header of the response is STATUS_INVALID_PARAMETER, this value MUST indicate the maximum number of bytes the server will accept to copy in a single request.

2.2.32.2 SRV_SNAPSHOT_ARRAY

The SRV_SNAPSHOT_ARRAY packet is returned to the client by the server in an SMB2 IOCTL Response for the FSCTL_SRV_ENUMERATE_SNAPSHOTS request, as specified in [MS-SMB] section 2.2.14.7.1. This packet MUST contain all the revision time-stamps that are associated with the Tree Connect share in which the open resides. This SRV_SNAPSHOT_ARRAY is placed in the **Buffer** field in the SMB2 IOCTL Response, and the **OutputOffset** and **OutputCount** fields MUST be updated to describe the buffer as specified in section 2.2.32. This packet consists of the following:



NumberOfSnapShots (4 bytes): The number of previous versions associated with the volume that backs this file.

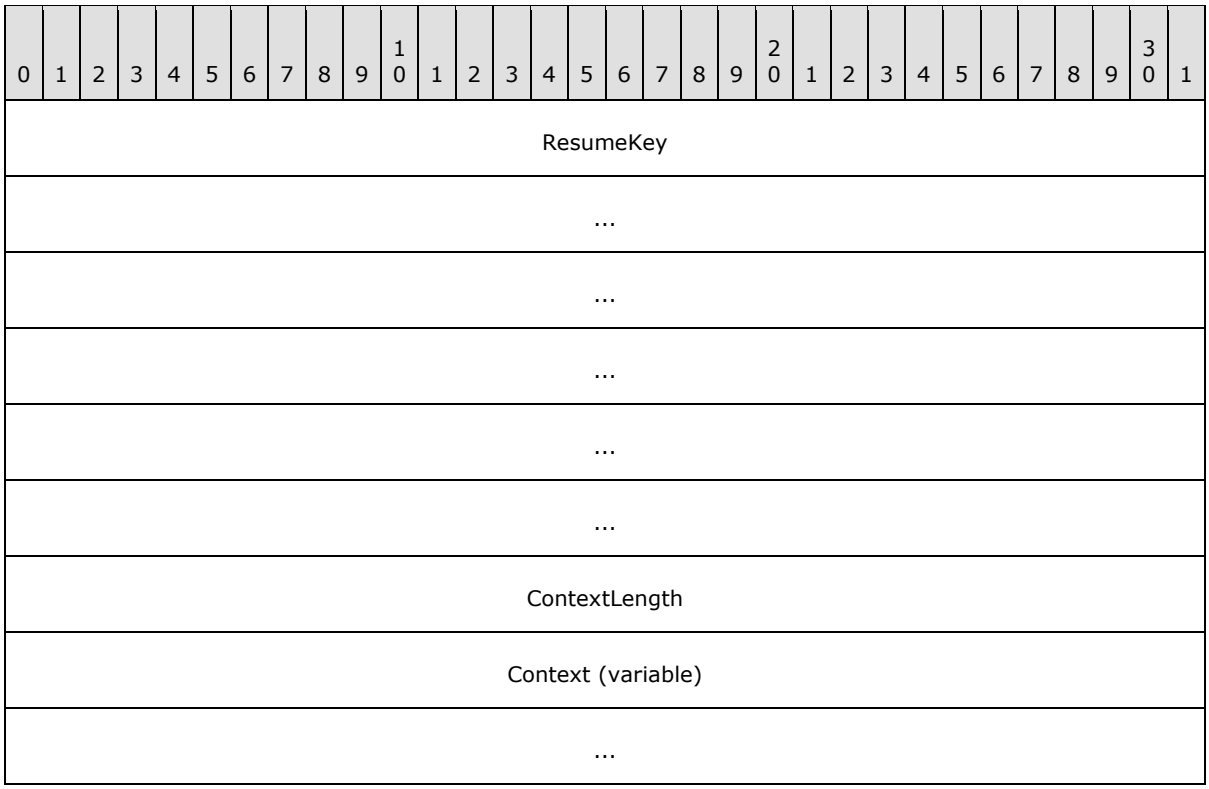
NumberOfSnapShotsReturned (4 bytes): The number of previous version time stamps returned in the **SnapShots**[] array.

SnapShotArraySize (4 bytes): This field specifies the length, in bytes, of the **SnapShots**[] array.

SnapShots (variable): An array of time stamps in GMT format, as specified by an **@GMT token**, which are separated by UNICODE NULL characters and terminated by two UNICODE NULL characters.

2.2.32.3 SRV_REQUEST_RESUME_KEY Response

The SRV_REQUEST_RESUME_KEY packet is returned to the client by the server in an SMB2 IOCTL Response for the **FSCTL_SRV_REQUEST_RESUME_KEY** request. This **SRV_REQUEST_RESUME_KEY** is placed in the **Buffer** field in the SMB2 IOCTL Response, and the **OutputOffset** and **OutputCount** fields MUST be updated to describe the buffer as specified in section 2.2.32. This packet consists of the following:



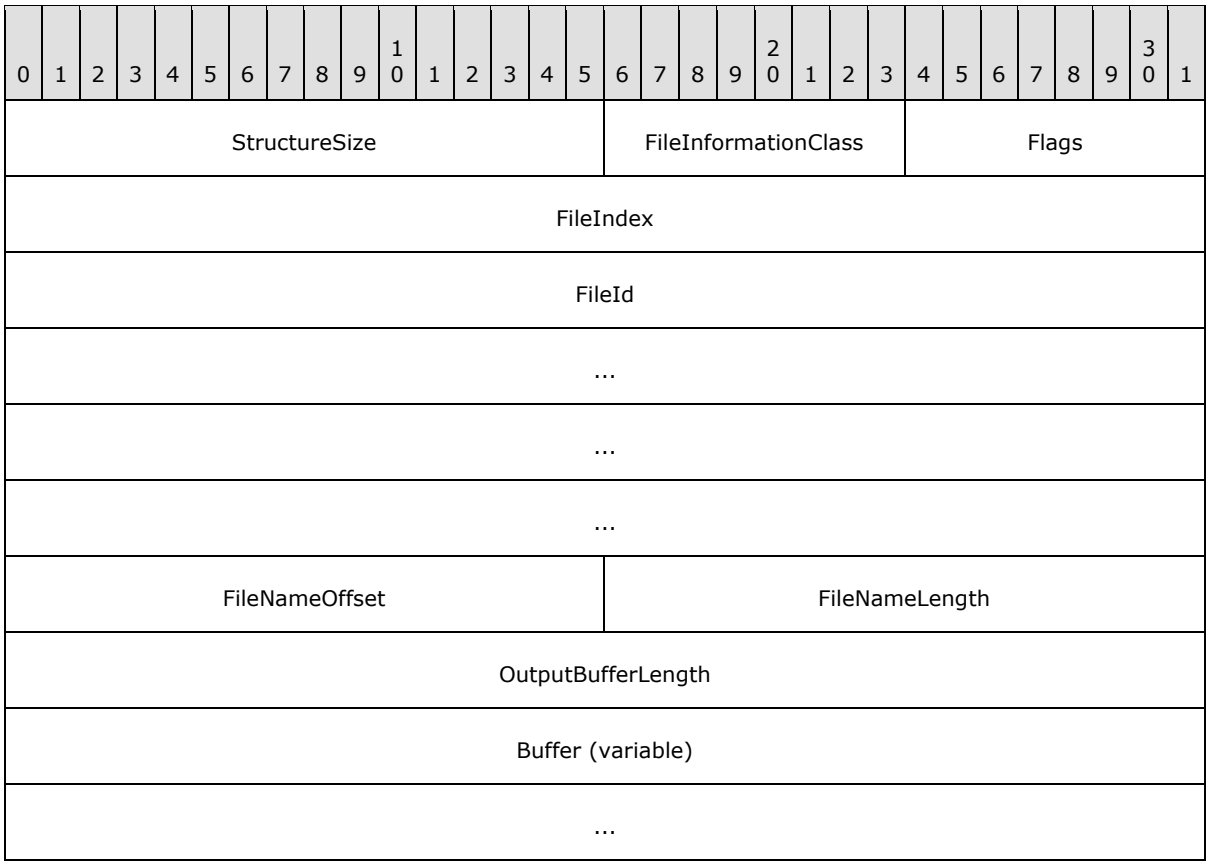
ResumeKey (24 bytes): A 24-byte resume key generated by the server that can be subsequently used by the client to uniquely identify the source file in an **FSCTL_SRV_COPYCHUNK** request. The resume key MUST be treated as a 24-byte opaque structure. The client that receives the 24-byte resume key MUST not attach any interpretation to this key and MUST treat it as an opaque value.

ContextLength (4 bytes): The length, in bytes, of the context information. This field is unused. The server SHOULD set this field to 0 when sending a response. This field MUST be ignored by the client when the message is received.

Context (variable): The context extended information. The length of **Context** MUST be greater than or equal to 4 bytes. If **ContextLength** is less than 4, the bytes in the **ContextLength** array that are not described by **ContextLength** SHOULD be initialized to 0. <28>

2.2.33 SMB2 QUERY_DIRECTORY Request

The SMB2 QUERY_DIRECTORY Request packet is sent by the client to obtain a directory enumeration on a directory open. This request consists of an SMB2 header, as specified in section 2.2.1, followed by this request structure:



StructureSize (2 bytes): The client MUST set this field to 33, indicating the size of the request structure, not including the header. The client MUST set this field to this value regardless of how long **Buffer**[] actually is in the request being sent.

FileInformationClass (1 byte): The file information class describing the format that data MUST be returned in. Possible values are as specified in [MS-FSCC] section 2.4. This field MUST contain one of the following values:

Value	Meaning
FileDirectoryInformation 0x01	Basic information about a file or directory. Basic information is defined as the file's name, time stamp, size and attributes. File attributes are as specified in [MS-FSCC] section 2.6.
FileFullDirectoryInformation 0x02	Full information about a file or directory. Full information is defined as all the basic information plus extended attribute size.
FileIdFullDirectoryInformation 0x26	Full information plus volume file ID about a file or directory. A volume file ID is defined as a unique number assigned to a given volume.
FileBothDirectoryInformation 0x03	Basic information plus extended attribute size and short name about a file or directory.
FileIdBothDirectoryInformation 0x25	FileBothDirectoryInformation plus volume ID about a file or directory.

Value	Meaning
FileNamesInformation 0x0C	Detailed information on the names of files in a directory.

Flags (1 byte): Flags indicating how the query directory operation MUST be processed. This field MUST be constructed using the following values:

Value	Meaning
SMB2_RESTART_SCANS 0x01	The server MUST restart the enumeration from the beginning, but the search pattern is not changed.
SMB2_RETURN_SINGLE_ENTRY 0x02	The server MUST only return the first entry of the search results.
SMB2_INDEX_SPECIFIED 0x04	The server SHOULD return entries beginning at the byte number specified by **FileIndex**.<29>
SMB2_REOPEN 0x10	The server MUST restart the enumeration from the beginning, and the search pattern MUST be changed to the provided value. This often involves silently closing and reopening the directory on the server side.

Any combination of the three values SMB2_RESTART_SCANS, SMB2_RETURN_SINGLE_ENTRY, and SMB2_REOPEN is valid. SMB2_REOPEN implies SMB2_RESTART_SCANS as well.

FileIndex (4 bytes): Index number received in a previous enumeration from where to resume the enumeration. This MUST be used if SMB2_INDEX_SPECIFIED is set in Flags.

FileId (16 bytes): An SMB2_FILEID identifier of the directory on which to perform the enumeration. This is returned from an SMB2 Create Request to open a directory on the server.

FileNameOffset (2 bytes): The offset, in bytes, from the beginning of the SMB2 header to the search pattern to be used for the enumeration. This field MUST be 0 if no search pattern is provided.

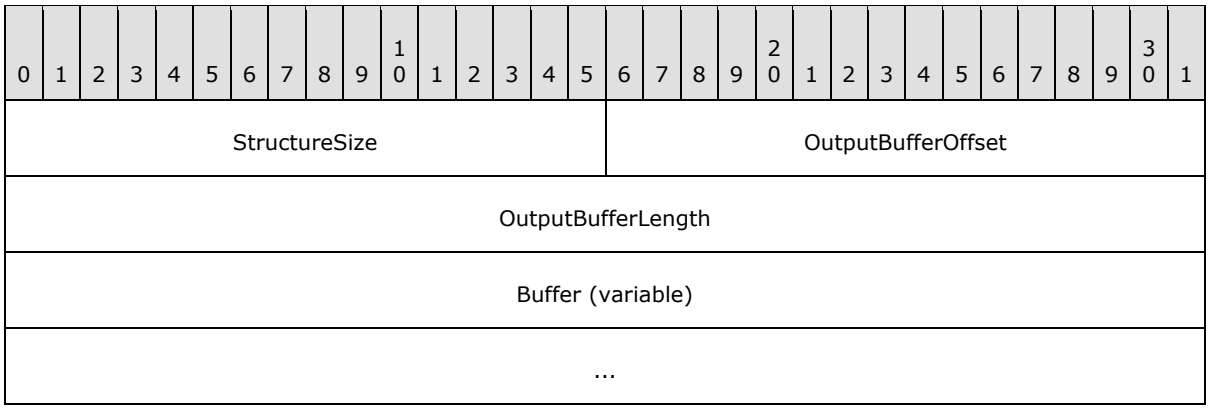
FileNameLength (2 bytes): The length, in bytes, of the search pattern. This field MUST be 0 if no search pattern is provided.

OutputBufferLength (4 bytes): The maximum number of bytes the server is allowed to return in the SMB2 QUERY_DIRECTORY Response.

Buffer (variable): A variable-length buffer containing the Unicode search pattern for the request, as described by the **FileNameOffset** and **FileNameLength** fields. The format, including wildcards and other conventions for this pattern, is specified in [CIFS] section 3.3.

2.2.34 SMB2 QUERY_DIRECTORY Response

The SMB2 QUERY_DIRECTORY Response packet is sent by a server in response to an SMB2 QUERY_DIRECTORY Request (section 2.2.33). This response consists of an SMB2 header, as specified in section 2.2.1, followed by this response structure:



StructureSize (2 bytes): The server MUST set this field to 9, indicating the size of the request structure, not including the header. The server MUST set this field to this value regardless of how long **Buffer**[] actually is in the request.

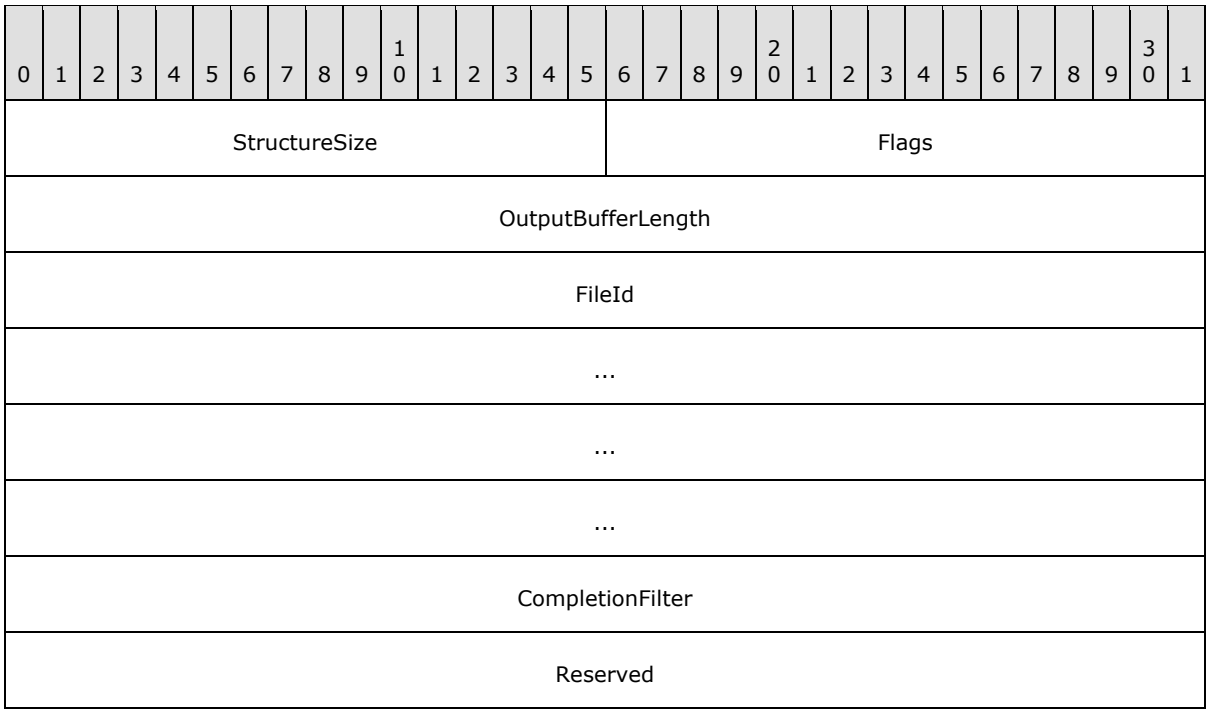
OutputBufferOffset (2 bytes): The offset, in bytes, from the beginning of the SMB2 header to the directory enumeration data being returned.

OutputBufferLength (4 bytes): The length, in bytes, of the directory enumeration being returned.

Buffer (variable): A variable-length buffer containing the directory enumeration being returned in the response, as described by the **OutputBufferOffset** and **OutputBufferLength**. The format of this content is as specified in [MS-FSCC] section 2.4, within the topic for the specific file information class referenced in the SMB2 QUERY_DIRECTORY request.

2.2.35 SMB2 CHANGE_NOTIFY Request

The SMB2 CHANGE_NOTIFY Request packet is sent by the client to request change notifications on a directory. This request consists of an SMB2 header, as specified in section 2.2.1, followed by this request structure:



StructureSize (2 bytes): The client MUST set this field to 32, indicating the size of the request structure, not including the header.

Flags (2 bytes): Flags indicating how the operation MUST be processed. This field MUST be constructed using the following values:

Value	Meaning
SMB2_WATCH_TREE 0x0001	The request MUST monitor changes on any file or directory contained beneath the directory specified by FileId.

OutputBufferLength (4 bytes): The maximum number of bytes the server is allowed to return in the SMB2 CHANGE_NOTIFY Response (section 2.2.36).

FileId (16 bytes): An SMB2_FILEID identifier of the directory to monitor for changes.

CompletionFilter (4 bytes): Specifies the types of changes to monitor. It is valid to choose multiple trigger conditions. In this case, if any condition is met, the client is notified of the change and the CHANGE_NOTIFY operation is completed. This field MUST be constructed using the following values:

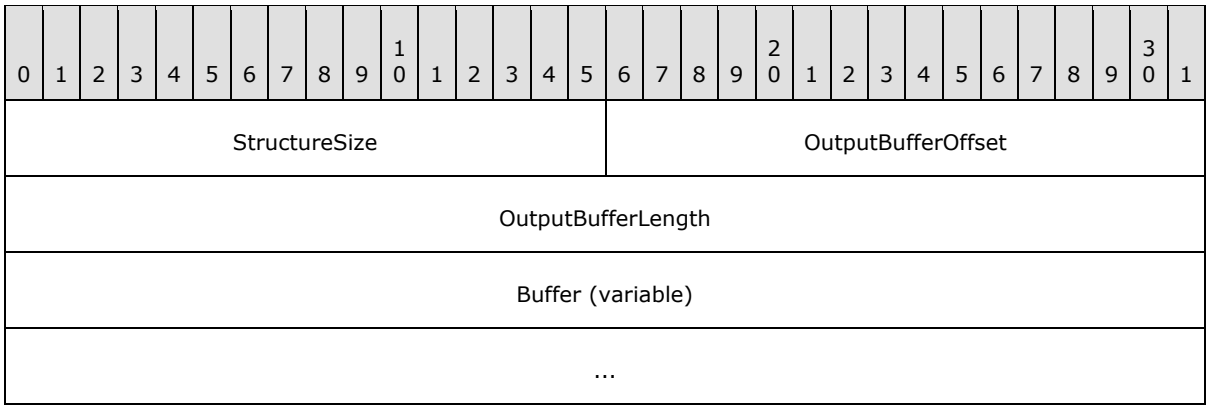
Value	Meaning
FILE_NOTIFY_CHANGE_FILE_NAME 0x00000001	The client is notified if a file-name changes.
FILE_NOTIFY_CHANGE_DIR_NAME 0x00000002	The client is notified if a directory name changes.
FILE_NOTIFY_CHANGE_ATTRIBUTES	The client is notified if a file's attributes change. Possible

Value	Meaning
0x00000004	file attribute values are specified in [MS-FSCC] 2.6.
FILE_NOTIFY_CHANGE_SIZE 0x00000008	The client is notified if a file's size changes.
FILE_NOTIFY_CHANGE_LAST_WRITE 0x00000010	The client is notified if the last write time of a file changes.
FILE_NOTIFY_CHANGE_LAST_ACCESS 0x00000020	The client is notified if the last access time of a file changes.
FILE_NOTIFY_CHANGE_CREATION 0x00000040	The client is notified if the creation time of a file changes.
FILE_NOTIFY_CHANGE_EA 0x00000080	The client is notified if a file's extended attributes (EAs) change.
FILE_NOTIFY_CHANGE_SECURITY 0x00000100	The client is notified of a file's access control list (ACL) settings change.
FILE_NOTIFY_CHANGE_STREAM_NAME 0x00000200	The client is notified if a **named stream** is added to a file.
FILE_NOTIFY_CHANGE_STREAM_SIZE 0x00000400	The client is notified if the size of a named stream is changed.
FILE_NOTIFY_CHANGE_STREAM_WRITE 0x00000800	The client is notified if a named stream is modified.

Reserved (4 bytes): Unused at present and MUST be treated as reserved. The client MUST set this field to 0, and the server MUST ignore it on receipt.

2.2.36 SMB2 CHANGE_NOTIFY Response

The SMB2 CHANGE_NOTIFY Response packet is sent by the server to transmit the results of a client's SMB2 CHANGE_NOTIFY Request (section 2.2.35). The server MUST send this packet only if a change occurs and MUST NOT send this packet otherwise. An SMB2 CHANGE_NOTIFY Request (section 2.2.35) will result in, at most, one response from the server. A server may choose to aggregate multiple changes into the same change notify response. The server MUST include at least one FILE_NOTIFY_INFORMATION structure if it detects a notification. This response consists of an SMB2 header, as specified in section 2.2.1, followed by this response structure:



StructureSize (2 bytes): The server MUST set this field to 9, indicating the size of the request structure, not including the header. The server MUST set the field to this value regardless of how long **Buffer**[] actually is in the request being sent.

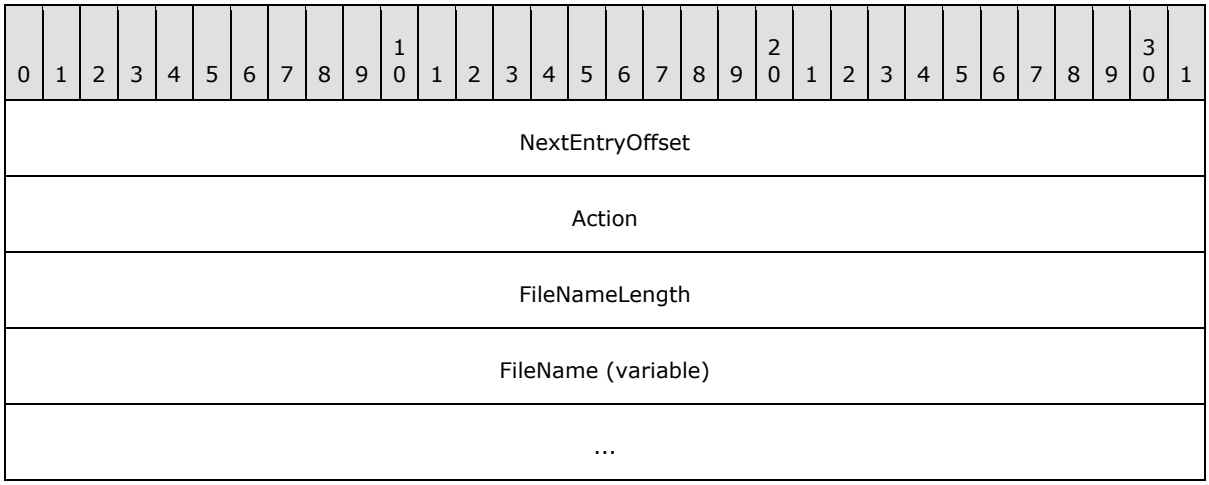
OutputBufferOffset (2 bytes): The offset, in bytes, from the beginning of the SMB2 header to the change information being returned.

OutputBufferLength (4 bytes): The length, in bytes, of the change information being returned.

Buffer (variable): A variable-length buffer containing the change information being returned in the response, as described by the **OutputBufferOffset** and **OutputBufferLength** fields. This field is an array of FILE_NOTIFY_INFORMATION.

2.2.36.1 FILE_NOTIFY_INFORMATION

The FILE_NOTIFY_INFORMATION packet is sent in the SMB2 CHANGE_NOTIFY Response (section 2.2.36) to return the changes that the client is being notified of. The structure consists of the following:



NextEntryOffset (4 bytes): The offset, in bytes, from the beginning of this structure to the subsequent FILE_NOTIFY_INFORMATION structure. If there are no subsequent structures, the **NextEntryOffset** field MUST be 0. **NextEntryOffset** MUST always be an integral multiple of

4. The **FileName** array MUST be padded to the next 4-byte boundary counted from the beginning of the structure.

Action (4 bytes): The changes that occurred on the file. This field MUST contain one of the following values.

Value	Meaning
FILE_ACTION_ADDED 0x00000001	The file was added to the directory.
FILE_ACTION_REMOVED 0x00000002	The file was removed from the directory.
FILE_ACTION_MODIFIED 0x00000003	The file was modified. This may be a change to the data or attributes of the file.
FILE_ACTION_RENAMED_OLD_NAME 0x00000004	The file was renamed, and this is the old name. If the new name resides within the directory being monitored, the client will also receive the FILE_ACTION_RENAMED_NEW_NAME as described below. If the new name resides outside of the directory being monitored, the client will not receive the FILE_ACTION_RENAMED_NEW_NAME.
FILE_ACTION_RENAMED_NEW_NAME 0x00000005	The file was renamed, and this is the new name. If the old name resides within the directory being monitored, the client will also receive the FILE_ACTION_RENAME_OLD_NAME. If the old name resides outside of the directory being monitored, the client will not receive the FILE_ACTION_RENAME_OLD_NAME.

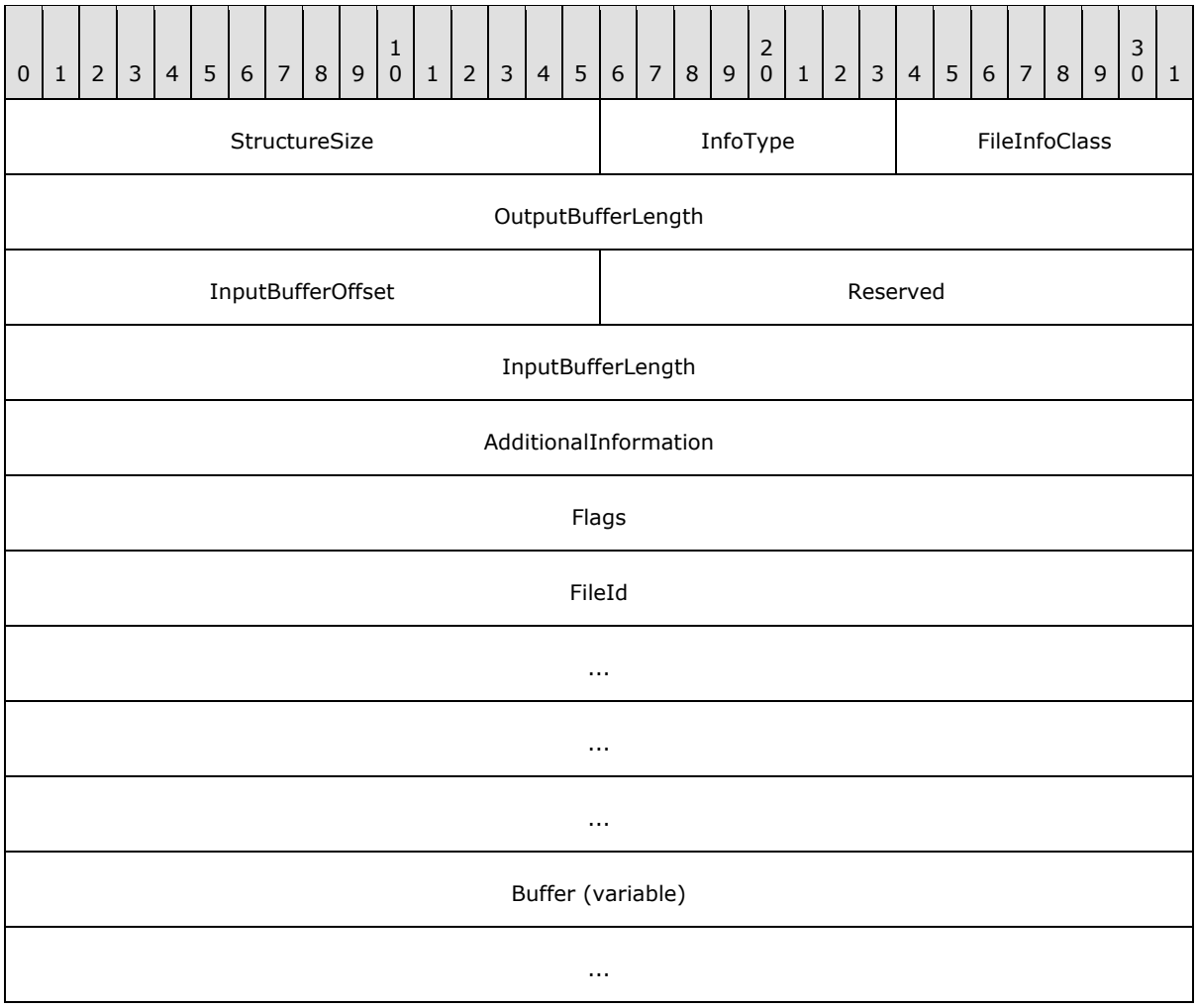
If two or more files have been renamed, then the corresponding FILE_NOTIFY_INFORMATION entries for each file rename MUST be consecutive in this response, in order for the client to make the correct correspondence between old and new names.

FileNameLength (4 bytes): The length, in bytes, of the file name in the **FileName**[] field.

FileName (variable): A Unicode string with the name of the file that changed.

2.2.37 SMB2 QUERY_INFO Request

The SMB2 QUERY_INFO Request (section 2.2.37) packet is sent by a client to request information on a file, named pipe, or underlying volume. This request consists of an SMB2 header, as specified in section 2.2.1, followed by this request structure:



StructureSize (2 bytes): The client MUST set this field to 41, indicating the size of the request structure, not including the header. The client MUST set this field to this value regardless of how long **Buffer**[] actually is in the request being sent.

InfoType (1 byte): The type of information queried. This field MUST contain one of the following values:

Value	Meaning
SMB2_0_INFO_FILE 0x01	The file information is requested.
SMB2_0_INFO_FILESYSTEM 0x02	The file system information is requested.
SMB2_0_INFO_SECURITY 0x03	The security information is requested.
SMB2_0_INFO_QUOTA 0x04	The file system quota information is requested.

FileInfoClass (1 byte): For file information queries, this field MUST contain one of the following FILE_INFORMATION_CLASS values, as specified in section 3.3.5.20.1 and in [MS-FSCC] section 2.4:

- FileBasicInformation
- FileStandardInformation
- FileInternalInformation
- FileEaInformation
- FileAccessInformation
- FilePositionInformation
- FileFullEaInformation
- FileModeInformation
- FileAlignmentInformation
- FileAllInformation
- FileAlternateNameInformation
- FileStreamInformation
- FilePipeInformation
- FilePipeLocalInformation
- FilePipeRemoteInformation
- FileCompressionInformation
- FileQuotaInformation
- FileNetworkOpenInformation
- FileAttributeTagInformation

For file system information queries, this field MUST contain one of the following FS_INFORMATION_CLASS values, as specified in section 3.3.5.20.2 and in [MS-FSCC] section 2.5:

- FileFsVolumeInformation
- FileFsSizeInformation
- FileFsDeviceInformation
- FileFsAttributeInformation
- FileFsFullSizeInformation
- FileFsObjectIdInformation

For quota and security queries, this field MUST be set to 0.

OutputBufferLength (4 bytes): The maximum number of bytes of information the server can send in the response.

InputBufferOffset (2 bytes): The offset to the input buffer from the beginning of the request. For quota requests, the input buffer MUST contain an SMB2_QUERY_QUOTA_INFO, as specified in section 2.2.37.1. For other information queries, this field MUST be set to 0.

Reserved (2 bytes): Unused at present and MUST be treated as reserved. The client MUST set this field to 0, and the server MUST ignore it on receipt.

InputBufferLength (4 bytes): The length of the input buffer. For quota requests, this MUST be the length of the contained SMB2_QUERY_QUOTA_INFO embedded in the request. For other information queries, this field MUST be set to 0.

AdditionalInformation (4 bytes): Provides additional information to the server.

If security information is being queried, this value contains a 4-byte bit field of flags indicating what security attributes MUST be returned. For more information about security descriptors, see **SECURITY_DESCRIPTOR** in [MS-DTYP].

Value	Meaning
OWNER_SECURITY_INFORMATION 0x00000001	The client is querying the owner from the security descriptor of the file or named pipe.
GROUP_SECURITY_INFORMATION 0x00000002	The client is querying the group from the security descriptor of the file or named pipe.
DACL_SECURITY_INFORMATION 0x00000004	The client is querying the discretionary access control list from the security descriptor of the file or named pipe.
SACL_SECURITY_INFORMATION 0x00000008	The client is querying the system access control list from the security descriptor of the file or named pipe.
LABEL_SECURITY_INFORMATION 0x00000010	The client is querying the integrity label from the security descriptor of the file or named pipe.

If **FileFullEaInformation** is being queried, this value MUST contain a ULONG indicating the index into the extended attribute information array at which to start the query.

For all other queries, this field MUST be set to 0.

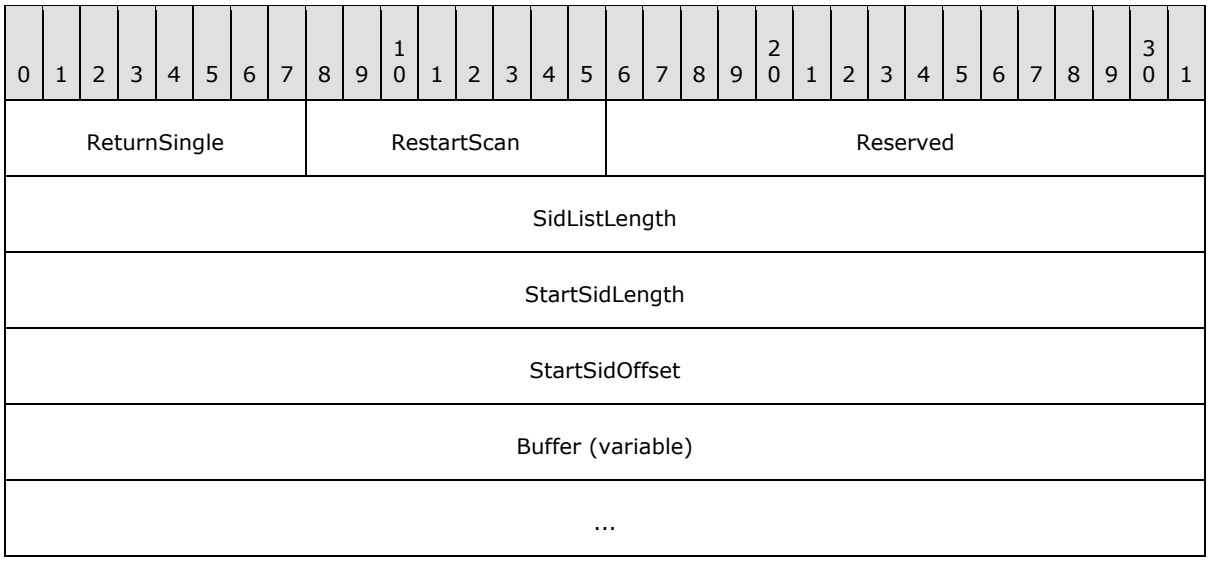
Flags (4 bytes): Unused at present and MUST be treated as reserved. The client MUST set this field to 0, and the server MUST ignore it on receipt.

FileId (16 bytes): An SMB2_FILEID identifier of the file or named pipe on which to perform the query. Queries for file system or quota information are directed to the volume on which the file resides.

Buffer (variable): A variable-length buffer containing the input buffer for the request, as described by the **InputBufferOffset** and **InputBufferLength** fields. If InputBufferLength is 0, then the client MUST set one byte of this field to 0, and the server MUST ignore it on receipt.

2.2.37.1 SMB2_QUERY_QUOTA_INFO

The SMB2_QUERY_QUOTA_INFO packet that specifies the quota information to return.



ReturnSingle (1 byte): A Boolean value. If the **ReturnSingle** field is any nonzero value, the server MUST return a single value. If zero, the server MUST return as many entries as will fit in the maximum output size that is indicated in the request.

RestartScan (1 byte): A Boolean value. If set, the quota information MUST be read from the beginning. If not set, the quota information MUST be continued from the previous enumeration that was executed on this open.

Reserved (2 bytes): Unused at present and MUST be treated as reserved. The client MUST set this field to 0, and the server MUST ignore it on receipt.

SidListLength (4 bytes): The length, in bytes, of the SidList, which is a linked list of FILE_GET_QUOTA_INFORMATION structures (see [MS-FSCC] section 2.4.32.1) that is pointed to by **StartSidOffset**. It MUST be set to zero if the buffer contains a single **SID** and does not contain a linked list of FILE_GET_QUOTA_INFORMATION structures.

StartSidLength (4 bytes): The length, in bytes, of a single SID. It MUST be set to zero if the **Buffer** field contains a linked list of FILE_GET_QUOTA_INFORMATION structures. See [MS-FSCC] section 2.4.32.1.<30>

StartSidOffset (4 bytes): The offset, in bytes, to a SID, from the beginning of the **Buffer** field. It MUST be zero if the buffer contains a linked list of FILE_GET_QUOTA_INFORMATION structures. See [MS-FSCC] section 2.4.32.1.

Buffer (variable): The format of the buffer is either a single SID, as specified in [MS-DTYP] section 2.4.2, or a linked list of one or more FILE_GET_QUOTA_INFORMATION structures. See [MS-FSCC] section 2.4.32.1.

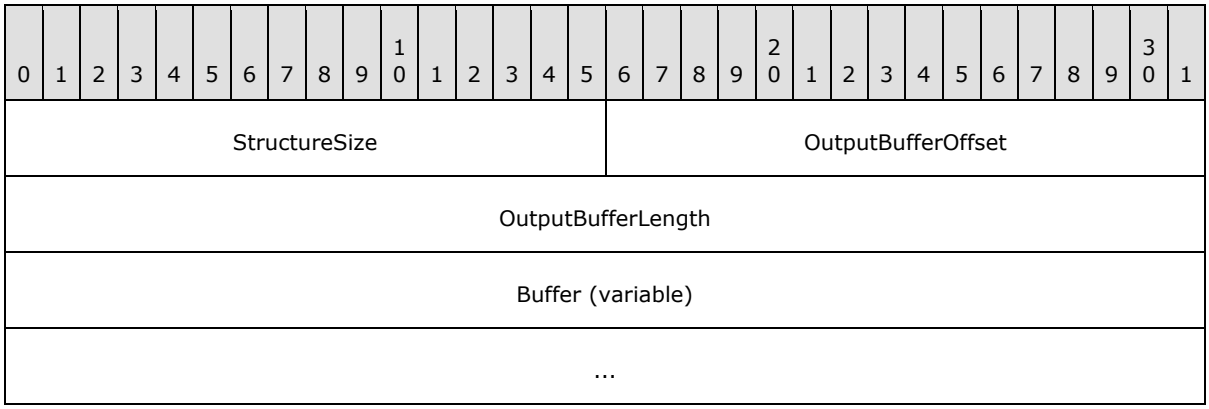
If the buffer contains a single SID, the **SidListLength** MUST be zero, the **StartSidLength** MUST be the length of the SID, and the **StartSidOffset** MUST be the offset to the SID.

If the buffer contains a single FILE_GET_QUOTA_INFORMATION structure, the **SidListLength** MUST be the length, in bytes, of the single FILE_GET_QUOTA_INFORMATION structure; the **StartSidLength** MUST be the length, in bytes, of a single SID that is contained in the FILE_GET_QUOTA_INFORMATION structure; and the **StartSidOffset** MUST be the offset to the single SID in the FILE_GET_QUOTA_INFORMATION structure.

If the buffer contains a linked list of FILE_GET_QUOTA_INFORMATION structures, the **SidListLength** MUST be the length, in bytes, of the list of FILE_GET_QUOTA_INFORMATION structures; the **StartSidLength** MUST be set to zero; and the **StartSidOffset** MUST be set to zero.

2.2.38 SMB2 QUERY_INFO Response

The SMB2 QUERY_INFO Response packet is sent by the server in response to an SMB2 QUERY_INFO Request packet. This response consists of an SMB2 header, as specified in section 2.2.1, followed by this response structure:



StructureSize (2 bytes): The server MUST set this field to 9, indicating the size of the request structure, not including the header. The server MUST set this field to this value regardless of how long **Buffer**[] actually is in the request being sent.

OutputBufferOffset (2 bytes): The offset, in bytes, from the beginning of the SMB2 header to the information being returned.

OutputBufferLength (4 bytes): The length, in bytes, of the information being returned.

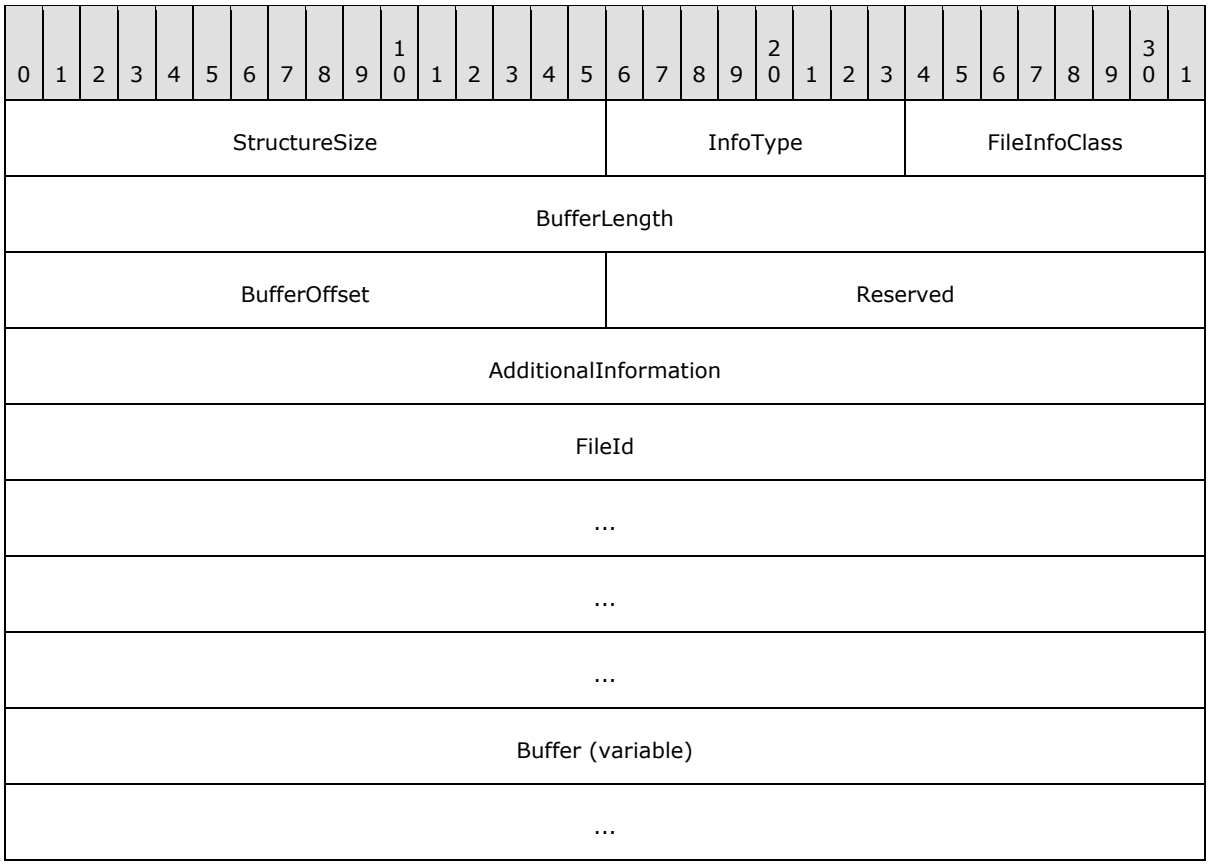
Buffer (variable): A variable-length buffer that contains the information that is returned in the response, as described by the **OutputBufferOffset** and **OutputBufferLength** fields. Buffer format depends on **InfoType** and **AdditionalInformation**, as follows:

InfoType	AdditionalInformation	Buffer format specification
SMB2_0_INFO_FILE	0	For FileFullEaInformation, the server MUST return as many whole contiguous elements as will fit in the Buffer starting at the index provided in the **AdditionalInformation** field of the request.
SMB2_0_INFO_FILESYSTEM	0	See [MS-FSCC] section 2.5.
SMB2_0_INFO_SECURITY	Any combination of the values: OWNER_SECURITY_INFORMATION GROUP_SECURITY_INFORMATION LABEL_SECURITY_INFORMATION DACL_SECURITY_INFORMATION	The security descriptor data structure, as specified in [MS-DTYP] section **2.4.6**, populated with the data specified by the AdditionalInformation value.

InfoType	AdditionalInformation	Buffer format specification
	SACL_SECURITY_INFORMATION	
SMB2_0_INFO_QUOTA	0	See [MS-FSCC] section 2.4.32.1.

2.2.39 SMB2 SET_INFO Request

The SMB2 SET_INFO Request packet is sent by a client to set information on a file or underlying file system. This request consists of an SMB2 header, as specified in section 2.2.1, followed by this request structure:



StructureSize (2 bytes): The client MUST set this field to 33, indicating the size of the request structure, not including the header. The client MUST set this field to this value regardless of how long Buffer[] actually is in the request being sent.

InfoType (1 byte): The type of information being set. The valid values are:

Value	Meaning
SMB2_0_INFO_FILE 0x01	The file information is being set.
SMB2_0_INFO_FILESYSTEM	The file system information is being set.

Value	Meaning
0x02	
SMB2_0_INFO_SECURITY 0x03	The security information is being set.
SMB2_0_INFO_QUOTA 0x04	The file system quota information is being set.

FileInfoClass (1 byte): This field MUST contain one of the following FILE_INFORMATION_CLASS values, as specified in section 3.3.5.20.1 and [MS-FSCC] section 2.4:

- FileBasicInformation
- FileRenameInformation
- FileLinkInformation
- FileDispositionInformation
- FilePositionInformation
- FileFullEaInformation
- FileModeInformation
- FileAllocationInformation
- FileEndOfFileInformation
- FilePipeInformation
- FileValidDataLengthInformation
- FileShortNameInformation

For setting file system information, this field MUST contain one of the following FS_INFORMATION_CLASS values, as specified in [MS-FSCC] section 2.5:

- FileFsControlInformation
- FileFsObjectIdInformation

For setting quota and security information, this field MUST be 0.

BufferLength (4 bytes): The length, in bytes, of the information to be set.

BufferOffset (2 bytes): The offset, in bytes, from the beginning of the SMB2 header to the information to be set.

Reserved (2 bytes): Unused at present and MUST be treated as reserved. The client MUST set this field to 0, and the server MUST ignore it on receipt.

AdditionalInformation (4 bytes): Provides additional information to the server.

If security information is being set, this value MUST contain a 4-byte bit field of flags indicating what security attributes MUST be applied. For more information about security descriptors, see [MS-DTYP] section 2.4.6.

Value	Meaning
OWNER_SECURITY_INFORMATION 0x00000001	The client is setting the owner in the security descriptor of the file or named pipe.
GROUP_SECURITY_INFORMATION 0x00000002	The client is setting the group in the security descriptor of the file or named pipe.
DACL_SECURITY_INFORMATION 0x00000004	The client is setting the discretionary access control list in the security descriptor of the file or named pipe.
SACL_SECURITY_INFORMATION 0x00000008	The client is setting the system access control list in the security descriptor of the file or named pipe.
LABEL_SECURITY_INFORMATION 0x00000010	The client is setting the integrity label in the security descriptor of the file or named pipe.

For all other set requests, this field MUST be 0.

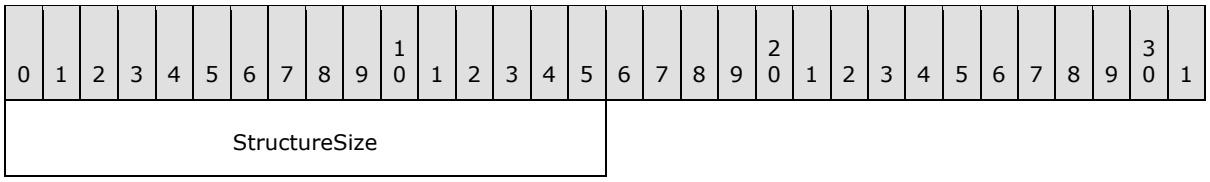
FileId (16 bytes): An SMB2_FILEID identifier of the file or named pipe on which to perform the set. Set operations for file system and quota information are directed to the volume on which the file resides.

Buffer (variable): A variable-length buffer that contains the information being set for the request, as described by the BufferOffset and BufferLength fields. Buffer format depends on **InfoType** and the **AdditionalInformation**, as follows:

InfoType	AdditionalInformation	Buffer format specification
SMB2_0_INFO_FILE	0	See [MS-FSCC] section 2.4
SMB2_0_INFO_FILESYSTEM	0	See [MS-FSCC] section 2.5
SMB2_0_INFO_SECURITY	Any combination of the values: OWNER_SECURITY_INFORMATION GROUP_SECURITY_INFORMATION LABEL_SECURITY_INFORMATION DACL_SECURITY_INFORMATION SACL_SECURITY_INFORMATION	The security descriptor data structure, as specified in [MS-DTYP] section 2.4.6, populated with the data specified by the AdditionalInformation value.
SMB2_0_INFO_QUOTA	0	See [MS-FSCC] section 2.4.32.1.

2.2.40 SMB2 SET_INFO Response

The SMB2 SET_INFO Response packet is sent by the server in response to an SMB2 SET_INFO Request (section 2.2.39) to notify the client that its request has been successfully processed. This response consists of an SMB2 header, as specified in section 2.2.1, followed by this response structure:



StructureSize (2 bytes): The server MUST set this field to 2, indicating the size of the request structure, not including the header.

3 Protocol Details

3.1 Common Details

3.1.1 Abstract Data Model

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

3.1.1.1 Global

The following global data is required by both the client and server:

RequireMessageSigning: A Boolean that, if set, indicates that this node requires that messages MUST be signed if the message is sent with a user security context that is neither anonymous nor guest. If not set, this node does not require that any messages be signed, but MAY still choose to do so if the other node requires it.

3.1.2 Timers

There are no timers common to both client and server.

3.1.3 Initialization

The value of RequireMessageSigning MUST be set based on system configuration, which is implementation-dependent.<31>

3.1.4 Higher-Layer Triggered Events

3.1.4.1 Signing An Outgoing Message

The client or server sending the message MUST provide the message length, the buffer containing the message, and the session key to use for signing. The following steps describe the signing process:

1. The sender MUST zero out the 16-byte signature field in the SMB2_HEADER of the message to be sent prior to generating the signature.
2. The sender MUST compute the SHA-256 hash employing the HMAC algorithm as specified in section 2 of [RFC2104] using the following parameters:
    1. The secret key is the 16-byte session key.
    2. The hashing algorithm is as specified in [FIPS180-2].
    3. The message from step 1 to be hashed.
3. The upper 16 bytes of the message hash generated by step 2 MUST be copied into the 16-byte signature field in the SMB2_HEADER.

3.1.5 Message Processing Events and Sequencing Rules

3.1.5.1 Verifying an Incoming Message

If a message is received and the SMB2_FLAGS_SIGNED bit is set in the SMB2_HEADER, the receiver MUST verify the signature of the message as specified below.

1. The receiver MUST look up the session using the SessionId in the SMB2 header of the incoming message, as specified in section 3.3.5.2.3, to fetch the 16-byte session key for use in step 4.
2. The receiver MUST save the 16-byte signature from the Signature field in the SMB2 header of the incoming message for use in step 6.
3. The receiver MUST zero out the 16-byte Signature field in the SMB2 header of the incoming message.
4. The receiver MUST compute the SHA-256 hash employing the HMAC algorithm as specified in section 2 of [RFC2104] using the following parameters:
    1. The secret key is the 16-byte session key obtained from step 1.
    2. The hashing algorithm is as specified in [FIPS180-2].
    3. The message from step 3 to be hashed.
5. The upper 16 bytes of the hash generated in step 4 is the expected signature. If this computed signature matches the saved signature from step 2, the receiver MUST continue to process the message. Otherwise, it MAY choose to return an error or disconnect the connection or discard the packet.<32>

If a message is received and the SMB2_FLAGS_SIGNED bit is not set in the SMB2_HEADER, but **Session.ShouldSign** is set to TRUE, the receiver MUST discard the packet and optionally, MAY disconnect the connection. If the SMB2_FLAGS_SIGNED bit is set, the validation MUST be performed as specified in the client/server-specific sections 3.2.5.1.2 and 3.3.5.2.1.

If a message is received, and the SMB2_FLAGS_SIGNED bit is not set in the SMB2_HEADER, and Session.ShouldSign is set to FALSE, signature verification MUST NOT be performed.

3.1.6 Timer Events

There are no timers common to both client and server.

3.1.7 Other Local Events

There are no local events common to both client and server.

3.2 Client Details

3.2.1 Abstract Data Model

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

3.2.1.1 Algorithm for Handling Available Message Sequence Numbers by the Client

The client MUST implement an algorithm to manage message **sequence numbers**. Sequence numbers are used to associate requests with responses and to determine what requests are allowed for processing. The algorithm MUST meet the following conditions:

- When the connection is first established, the allowable sequence numbers for sending a request MUST be set to the set { 0 }.
- The client MUST never send a request on a given connection with a sequence number that has already been used unless it is a request to cancel a previously sent request.
- The client MUST grow the set in a monotonically increasing manner based on the credits granted. If the set is { 0 }, and 2 credits are granted, the set MUST grow to { 0, 1, 2 }.
- The client SHOULD use the lowest available sequence number in its allowable set for each request.
- If a cancel request is sent, a client MUST NOT consume a sequence number. Otherwise, the client MUST consume a sequence number when it sends out an SMB2 request.

For the server side of this algorithm, see section 3.3.1.2.

3.2.1.2 Global

ConnectionTable: A table of active SMB2 transport connections, as specified in section 3.2.1.3, that are established to remote servers. The table MUST be indexed by the textual server name that is provided by the client, which is a DNS name, NetBIOS name, or IP address.

GlobalOpenTable: A table of the active opens to remote files or named pipes across all connections. The structure of an open is described in section 3.2.1.6. The table MUST be uniquely indexed by **Open.ClientLocalId** and by the persistent portion of **Open.FileId**.

3.2.1.3 Per SMB2 Transport Connection

Connection.SessionTable: A table of authenticated sessions, as specified in section 3.2.1.4, that the client has established on this SMB2 transport connection. The table MUST allow lookup by both **Session.SessionId** and by the security context of the user that established the connection.

Connection.OutstandingRequests: A table of requests that have been issued on this connection and are awaiting a response. The table MUST allow lookup by **MessageId**, and each request MUST store a time stamp of when the request was sent.

Connection.SequenceWindow: A table of available sequence numbers for sending requests to the server, as specified in section 3.2.1.1.

Connection.OpenTable: A table of opens, as specified in section 3.2.1.6. The table MUST allow lookup by either file name or by **Open.FileId**.

Connection.GSSNegotiateToken: A byte array containing the token received during a negotiation and remembered for authentication.

Connection.MaxTransactSize: The maximum buffer size, in bytes, that the server will accept on this connection for QUERY_INFO, QUERY_DIRECTORY, SET_INFO and CHANGE_NOTIFY operations. This field is applicable only for buffers sent by the client in SET_INFO requests, or returned from the server in QUERY_INFO, QUERY_DIRECTORY, and CHANGE_NOTIFY responses. <33>

Connection.MaxReadSize: The maximum read size, in bytes, that the server will accept in an SMB2 READ request on this connection.

Connection.MaxWriteSize: The maximum write size, in bytes, that the server will accept in an SMB2 WRITE request on this connection.

Connection.ServerGuid: A globally unique identifier that is generated by the remote server to uniquely identify the remote server. This field MUST NOT be used by a client as a secure method of identifying a server.

Connection.RequireSigning: A Boolean indicating whether the server requires all requests/responses on this connection to be signed.

3.2.1.4 Per Session

Session.SessionId: An 8-byte identifier returned by the server to identify this session on this SMB2 transport connection.

Session.TreeConnectTable: A table of tree connects, as specified in section 3.2.1.5. The table MUST allow lookup by either **TreeConnect.TreeConnectId** or by share name.

Session.SessionKey: The first 16 bytes of the cryptographic key for this authenticated context. If the cryptographic key is less than 16 bytes it is right padded with zero bytes.

Session.ShouldSign: A Boolean that indicates whether this session MUST sign communication if signing is enabled on this connection.

Session.Connection: A reference to the connection on which this session was established.

Session.SecurityContext: An OS-specific security context of the user who initiated the session.

3.2.1.5 Per Tree Connect

TreeConnect.TreeConnectId: A 4-byte identifier returned by the server to identify this tree connect.

TreeConnect.MaximalAccess: The maximal rights that the security principal that is described by **TreeConnect.Session** has on the target share. This MUST be an access mask value, as specified in section 2.2.13.1.

TreeConnect.Session: A reference to the session on which this tree connect was established.

3.2.1.6 Per Open

Open.ClientLocalId: An identifier generated by the client to return to higher-level applications to allow them to reference this open.

Open.FileId: The **SMB2_FILEID**, as specified in section 2.2.14.1, returned by the server for this open.

Open.MaximalAccess: The maximal rights that the security principal, as specified in **Open.TreeConnect.Session**, has on the file or named pipe. This MUST be an access mask value, as specified in section 2.2.13.1.

Open.TreeConnect: A reference to the tree connect on which this Open was established.

Open.Connection: A reference to the SMB2 transport connection on which this open was established.

Open.OplockLevel: The current oplock level for this open.

Open.Durable: A Boolean that indicates whether this open is setup for reestablishment after a disconnect.

Open.FileName: A Unicode string with the name of the file.

3.2.2 Timers

3.2.2.1 Request Expiration Timer

This timer optionally regulates the amount of time the client waits for a response from the server before failing the request and disconnecting the connection. The client MAY choose to implement this timer.<34>

3.2.2.2 Idle Connection Timer

The client MAY choose to scan existing connections on a periodic basis, and disconnect connections on which no opens exist and no operations have been issued for a certain period of time.<35>

3.2.3 Initialization

ConnectionTable is set to an empty table.

GlobalOpenTable is set to an empty table.

Connection.RequireSigning MUST be set to **FALSE** for each connection that is created.

3.2.4 Higher-Layer Triggered Events

The SMB2 client protocol is initiated and subsequently driven by a series of higher-layer triggered events in the following categories:

- Initiating a connection to a remote share.
- Opening a file, named pipe, or directory on a remote share.
- Accessing a file, named pipe, or directory on a remote share (reading, writing, locking, unlocking, handling IOCTL requests, querying or applying attributes, and so on).
- Closing a file, named pipe, or directory on a remote share.
- Closing a connection to a remote share.
- Required actions for sending any outgoing message.
- Requests for the session key for authenticated sessions.

The following sections provide details on the above events.

3.2.4.1 Sending Any Outgoing Message

Unless specifically noted in a subsequent section, the following logic MUST be applied to any request message being sent from the client to the server.

3.2.4.1.1 Signing the Message

If the request message being sent contains a **SessionId** that is nonzero in the SMB2 header field, and the session in Connection.SessionTable identified by that identifier has **Session.ShouldSign** equal to TRUE, the client MUST sign the request as specified below. Otherwise, the packet MUST NOT be signed.

The client MUST sign the request as specified in section 3.1.4.1. The client provides the **Session.SessionKey**, the length of the request, and the request itself, and calculates the signature as described.

3.2.4.1.2 Requesting Credits from the Server

The client MAY request more credits if it wants to have more outstanding simultaneous requests. If the client is requesting more credits, it MUST set **CreditsRequested** equal to the number of credits it is requesting from the server. If the client does not want to request more credits, it MUST set the number of credits it is requesting to 1.<36>

3.2.4.1.3 Associating the Message with a MessageId

Any message sent from the client to the server MUST have a valid MessageId placed in the SMB2 header. The client MUST take an available identifier from **Connection.SequenceWindow**. If there are no available identifiers, the request MUST wait for an available identifier before it can be sent to the server. For a description of how the available credit window is changed, see sections 3.2.1.1 and 3.3.1.2. The request MUST be inserted into **Connection.OutstandingRequests**. If the client chooses to implement the Request Expiration Timer, the client MUST then set the Request Expiration Timer to signal at the configured time-out interval for this command.

3.2.4.1.4 Sending Compounded Requests

A non-zero value for the **NextCommand** field in the SMB2 header indicates a compound request. **NextCommand** in the SMB2 header of a request specifies an offset, in bytes, from the beginning of the SMB2 header under consideration to the start of the 8-byte aligned SMB2 header of the subsequent request. Such compounding MAY be used to append multiple requests up to the maximum size that is supported by the transport. The client MUST choose one of two possible styles of message compounding specified in subsequent sections. These two styles MUST NOT be intermixed in the same transport send. <37> Compounded requests MUST be aligned on 8 byte boundaries. If a client or server receives a message that is not aligned on such a boundary, the machine SHOULD disconnect the connection. <38>

3.2.4.1.4.1 Compounding Unrelated Requests

The SMB2_FLAGS_RELATED_OPERATIONS MUST not be set in the flags of all SMB2 Headers in the chain. The client MUST NOT expect the responses of unrelated requests to arrive in the same transport receive from the server, or even in the same order they were sent.<39>

3.2.4.1.4.2 Compounding Related Requests

The SMB2_FLAGS_RELATED_OPERATIONS MUST be set in the flags of SMB2 Headers on all requests except the first one. Theclient MAY choose to send multiple requests required to perform a desired action as a compounded send containing related operations. Two examples would be to open a file and read from it, or to write to an open and close it. This form of compounding MUST NOT be used in combination with compounding unrelated requests within a single send.<40>

To issue a compounded send of related requests, take the following steps:

- The client MUST construct the initial request as it would if sending the requests separately.
- It MUST set the NextCommand in the SMB2 header of the initial request to the offset, in bytes, from the beginning of the SMB2 header to the beginning of the 8-byte aligned SMB2 header of the subsequent request. It MUST NOT set the SMB2_FLAGS_RELATED_OPERATIONS flag in the Flags field of the SMB2 header for this request.
- The client MUST construct the subsequent request as it would do normally. For any subsequent requests the client MUST set SMB2_FLAGS_RELATED_OPERATIONS flag in the Flags field of the SMB2 header. If the client wishes to use either the **SessionID**, **TreeID** or **FileID** fields of the previous request, or generated as a result of processing the previous request, the client MUST set the corresponding field to -1.
- The client MUST NOT create a compounded series of requests such that the combined length of the request exceeds the maximum send size that is supported by the underlying transport.

3.2.4.2 Application Requests a Connection to a Share

The higher-layer application MUST provide:

- The name of the server to connect to.
- The name of the share to connect to.
- The security principal that is attempting the connection.

The client MUST follow the steps outlined in the following flow chart. The request to connect to a server could be either explicit (the application requests the connection directly) or implicit (the application requests opening a file with a network path including server and share). In either case, the steps described in the flow chart MUST be followed. For the implicit case, any error returned from the connection attempt MUST be returned as the error code for the operation that initiated the implicit connection attempt. For the explicit case, any error returned from the connection attempt MUST be returned to the calling application.

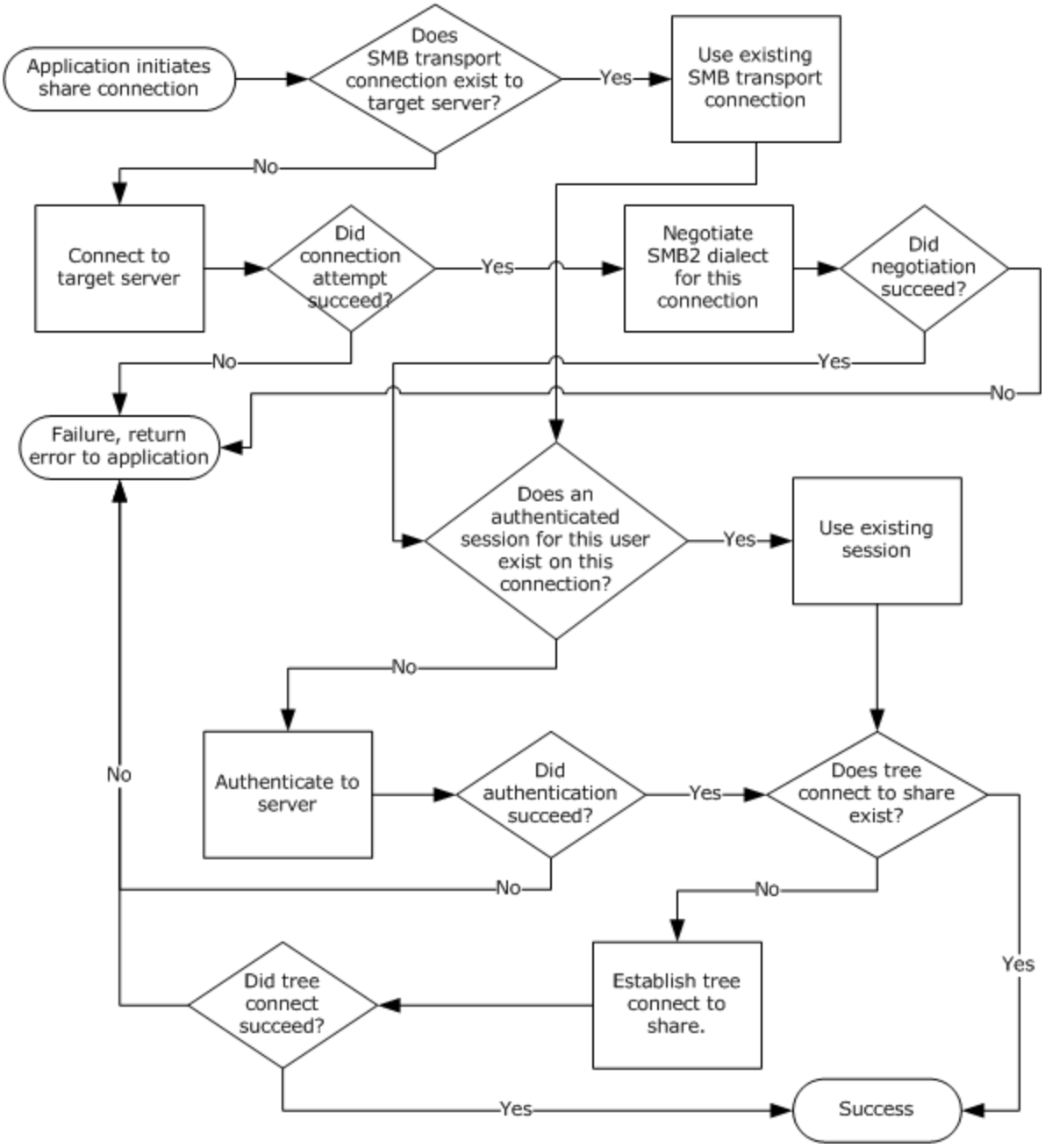


Figure 1: Client MUST follow this chart

3.2.4.2.1 Connecting to the Target Server

The client MAY search ConnectionTable for an existing connection to the target server. Otherwise, the client MAY choose to establish a new connection to the target server. The client MAY reuse the same connection and session for a specific security principal that is passed in from the higher-layer application. If an existing connection is found, it MAY be used, and processing continues, as specified in section 3.2.4.2.3.<41>

If an existing connection is not found, the client MUST attempt to connect to the target server over the registered transports specified in section 2.1 and [MS-SMB] section 2.1. If the connection attempt is successful, a connection object MUST be created, as specified in section 3.2.1.3, with the following default parameters:

- **Connection.SessionTable** MUST be set to an empty table.
- **Connection.OutstandingRequests** MUST be set to an empty table.
- **Connection.SequenceWindow** MUST be set to a sequence window, as specified in section 3.2.1.1, with a single starting sequence number available, which is "0".
- **Connection.OpenTable** MUST be set to an empty table.
- **Connection.GSSNegotiateToken** MUST be set to an empty array.

This connection MUST be inserted into ConnectionTable, and processing MUST continue, as specified in section 3.2.4.2.2.

If the connection attempt fails, the client returns the error code to the calling application.

3.2.4.2.2 Negotiating the Protocol

When a new connection is established, the client MUST negotiate capabilities with the server. The client MAY use either of two possible methods for negotiation.<42>

The first is a multi-protocol negotiation that involves sending an SMB message to negotiate the use of SMB2. If the server does not support the SMB 2.0 Protocol, this method allows the negotiation to fall back to older SMB dialects, as specified in [MS-SMB].

The second method is to send an SMB2-only negotiate message. This method will result in successful negotiation only for servers that support the SMB 2.0 Protocol.

3.2.4.2.2.1 Multi-Protocol Negotiate

To negotiate either the SMB 2.0 Protocol or the SMB Protocol in a single request, the client MUST allocate sequence number 0 from Connection.SequenceWindow. It MUST construct an SMB_COM_NEGOTIATE message following the syntax as specified in [MS-SMB] sections 2.2.2 and 3.2.4.2, and in [CIFS] section 4.1.1. The client MUST include the dialect string "SMB 2.002"<43> in the list of dialects, along with any SMB dialects that it supports. The remaining field in the request MUST be set up as specified in [MS-SMB] section 3.2.4.2.

3.2.4.2.2.2 SMB2-Only Negotiate

To issue an SMB2-only negotiate, the client MUST construct an SMB2 NEGOTIATE request following the syntax as specified in section 2.2.3:

- Allocate sequence number 0 from the Connection.SequenceWindow and place it in the MessageId field of the SMB2 header.
- Set the Command field in the SMB2 header to SMB2 NEGOTIATE.
- Set ProcessId to 0xFEFF.
- Set DialectCount to 1.
- Set the first value in the Dialects[] array to 0.

- Set SMB2_NEGOTIATE_SIGNING_ENABLED to TRUE in SecurityMode.
- If RequireMessageSigning is TRUE, the client MUST set SMB2_NEGOTIATE_SIGNING_REQUIRED to TRUE in SecurityMode.
- The client MAY set Capabilities, ClientGuid, and ClientStartTime to appropriate values.<44>

This request MUST be sent to the server.

3.2.4.2.3 Authenticating the User

The client MUST search Connection.SessionTable for an existing session for the security principal that is provided in the request to connect to a share. If a session is found, processing the request to connect to a share MUST continue as specified in section 3.2.4.2.4.

The client MAY either:<45>

- Pass the **Connection.GSSNegotiateToken** to the configured GSS authentication mechanism to obtain a GSS output token for the authentication protocol exchange, as specified in [MS-SPNG] section 3.3.5.1.

  OR

- Choose to ignore the **Connection.GSSNegotiateToken** that is received from the server, and initiate a normal GSS sequence, as specified in [MS-SPNG] section 3.3.4 and [RFC4178] section 3.2.

In either case, it MUST call the GSS authentication protocol with the MutualAuth and Delegate options. In addition the client MUST also set the GSS_C_FRAGMENT_TO_FIT parameter as specified in [MS-SPNG] section 3.1.1. The GSS-API output token is up to a size limit determined by local policy <46> when GSS_C_FRAGMENT_TO_FIT is set.

If the GSS authentication protocol returns an error, the share connect attempt MUST be aborted and the error MUST be returned to the higher-level application.

If the GSS authentication succeeds, the client MUST construct an SMB2 SESSION_SETUP Request, as specified in section 2.2.5. The SMB2 header MUST be initialized as follows:

- The **Command** field MUST be set to SMB2 SESSION_SETUP.
- A sequence number is allocated from **Connection.SequenceWindow**. If no sequence number is available, it MUST wait for one to become available. The client sets **MessageId** to the sequence number received from **Connection.SequenceWindow**.
- The **ProcessId** field MUST be set to 0xFEFF.

The SMB2 SESSION_SETUP request MUST be initialized as follows:

- The SMB2_NEGOTIATE_SIGNING_ENABLED bit in the **SecurityMode** field SHOULD be set.<47>
- If RequireMessageSigning is TRUE, the SMB2_NEGOTIATE_SIGNING_REQUIRED bit in the **SecurityMode** field MUST be set.
- The **VcNumber** field MUST be set to 0.
- If the client supports the Distributed File System (DFS), as specified in [MS-DFSC], the SMB2_GLOBAL_CAP_DFS bit in the **Capabilities** field MUST be set.

- The GSS output token is copied into the **Buffer** field in the request. The client MUST set **SecurityBufferOffset** and **SecurityBufferLength** to describe the location and length of the GSS output token in the request.

This request MUST be sent to the server.

3.2.4.2.3.1 Application Requests Reauthenticating a User

It is possible that the server indicates that authentication has expired, as specified in sections 3.3.5.7 and 3.3.5.9, and the application or the client itself requests that an existing session be reauthenticated. To do so, the client MUST issue a subsequent session setup request for the SessionId of the session being reauthenticated.

The client MAY either:<48>

- Pass the Connection.GSSNegotiateToken to the configured GSS authentication mechanism to obtain a GSS output token for the authentication protocol exchange, as specified in [MS-SPNG] section 3.3.5.1.

  or

- Choose to ignore the Connection.GSSNegotiateToken received from the server, and initiate a normal GSS sequence as specified in [MS-SPNG] section 3.3.4 and [RFC4178] section 3.2.

In either case, it initializes the GSS authentication protocol with the MutualAuth and Delegate options. In addition the client MUST also set the GSS_C_FRAGMENT_TO_FIT parameter as specified in [MS-SPNG] section 3.1.1. The GSS-API output token is up to a size limit determined by local policy <49> when GSS_C_FRAGMENT_TO_FIT is set .

If the GSS authentication protocol returns an error, the reauthentication attempt MUST be aborted, and the error MUST be returned to the higher-level application.

If the GSS authentication succeeds, the client MUST construct an SMB2 SESSION_SETUP request, as specified in section 2.2.5. The SMB2 header MUST be initialized as follows:

- The **Command** field MUST be set to SMB2 SESSION_SETUP.
- A sequence number is allocated from **Connection.SequenceWindow**. If no sequence number is available, it MUST wait for one to become available. The client sets MessageId to the sequence number that is received from **Connection.SequenceWindow**.
- The **ProcessId** field MUST be set to 0xFEFF.
- The **SessionId** field MUST be set to the **Session.SessionId** for the session being reauthenticated.

The SMB2 SESSION_SETUP request MUST be initialized as follows:

- The SMB2_NEGOTIATE_SIGNING_ENABLED bit in the **SecurityMode** field SHOULD be set.<50>
- If RequireMessageSigning is TRUE, the SMB2_NEGOTIATE_SIGNING_REQUIRED bit in the **SecurityMode** field MUST be set.
- The **VcNumber** field MUST be set to 0.
- If the client supports the Distributed File System (DFS), as specified in [MS-DFSC], the SMB2_GLOBAL_CAP_DFS bit in the **Capabilities** field MUST be set.

- The GSS output token is copied into the Buffer field in the request. The client MUST set **SecurityBufferOffset** and **SecurityBufferLength** to describe the location and length of the GSS output token in the request.

This request MUST be sent to the server.

3.2.4.2.4 Connecting to the Share

The client MUST search **Session.TreeConnectTable** for an existing tree connect to the share being connected to. If a tree connect is found, the client MUST complete the request to connect to the share successfully.

If a tree connect is not found, the client MUST construct an SMB2 TREE_CONNECT Request using the syntax specified in section 2.2.9. The SMB2 header MUST be initialized as follows:

- The Command field is set to SMB2 TREE_CONNECT.
- A sequence number is allocated from Connection.SequenceWindow. If no sequence number is available, it MUST wait for one to become available. The client sets MessageId to the sequence number received from **Connection.SequenceWindow**.
- The **ProcessId** field is set to 0xFEFF.
- The **SessionId** field is set to Session.SessionId of the session identified or established as part of processing section 3.2.4.2.3.

The SMB2 TREE_CONNECT Request MUST be initialized as follows:

- The target share path, including server name, in the format "\\server\share", is copied into the **Buffer** field of the request. **PathOffset** and **PathLength** MUST be set to describe the location and length of the target share path in the request.

This request MUST be sent to the server.

3.2.4.3 Application Requests Opening a File

The application MUST provide:

- The name of the server on which to open the file.
- The name of the share on which to open the file.
- The security principal of the user performing the open.
- The share relative path to the file being opened.
- The wanted access for the open, as specified in section 2.2.13.1.
- The sharing mode for the open, as specified in section 2.2.13.
- The create options to be applied for the open, as specified in section 2.2.13.
- The create disposition for the open, as specified in section 2.2.13.
- The file attributes for the open, as specified in section 2.2.13.
- The impersonation level for the open, as specified in section 2.2.13 (optional).

- The security flags for the open, as specified in section 2.2.13 (optional).
- The requested oplock level for the open, as specified in section 2.2.13 (optional).
- As outlined in subsequent sections, the application may also provide a series of create contexts, as specified in section 2.2.13.2.

The client MUST create a connection to the target share using the server name, the share name, and the security principal, as specified in section 3.2.4.2. If this fails, the client MUST return this error code to the calling application. If this succeeds, the TreeConnect MUST be used in the remaining processing that is described below.

The client MUST construct an SMB2 CREATE Request using the syntax specified in section 2.2.13. The SMB2 header MUST be initialized as follows:

- The **Command** field is set to SMB2 CREATE.
- A sequence number is allocated from **Connection.SequenceWindow**. If no sequence number is available, it MUST wait for one to become available. The client sets MessageId to the sequence number that is received from **Connection.SequenceWindow**.
- The **SessionId** field is set to **TreeConnect.Session.SessionId**.
- The **TreeId** field is set to **TreeConnect.TreeConnectId**.
- The **ProcessId** field is set to a 4-byte identifier for the process that issued the create request.

The SMB2 CREATE Request MUST be initialized as follows:

- The **SecurityFlags** field is set to 0.
- The **RequestedOplockLevel** field is set to the oplock level that is requested by the application. If the application does not provide a requested oplock level, the client MUST choose an appropriate oplock level.<51>
- The **ImpersonationLevel** field is set to the application-provided impersonation level. If the application did not provide an impersonation level, the client sets the **ImpersonationLevel** to **Impersonation**.
- The client sets the **DesiredAccess** field to the value that is provided by the application.
- The client sets the **FileAttributes** field to the attributes that are provided by the application.
- The client sets the **ShareAccess** field to the sharing mode that is provided by the application.
- The client sets the **CreateDisposition** field to the create disposition that is provided by the application.
- The client sets the **CreateOptions** field to the create options that are provided by the application.
- The client copies the relative path into the **Buffer**, and sets the **NameLength** to the length, in bytes, of the relative path and the **NameOffset** to the offset, in bytes, to the relative path from the beginning of the SMB2 header.
- The client copies any provided create contexts into the **Buffer** after the file name, and sets the **CreateContextOffset** to the offset, in bytes, to the create contexts from the beginning of the SMB2 header and sets the **CreateContextLength** to the length, in bytes, of the array of create

contexts. If there are no provided create contexts, CreateContextLength and CreateContextOffset MUST BE set to 0.

This request MUST be sent to the server.

3.2.4.3.1 Application Requests Opening a Named Pipe

For opening a named pipe, the application provides the same parameters that are specified in section 3.2.4.3, except the name of the share MUST be "IPC$". This share name indicates that the open targets a named pipe.

3.2.4.3.2 Application Requests Sending a File to Print

For sending a file to a printer, the application opens the root of a print share, writes data, and closes the file. The semantics and parameters are the same as specified in section 3.2.4.3, except that the name of the share MUST be the name of a printer share, and the share relative path must be NULL.

3.2.4.3.3 Application Requests Creating a File with Extended Attributes

To create a file with extended attributes, in addition to the parameters that are specified in section 3.2.4.3, the application MUST provide a buffer of extended attributes in the format that is specified in [MS-FSCC] section 2.4.15. The client MUST construct a create context, as specified in section 2.2.13.2.1, and append it to any other create contexts being issued and pass it in for create processing.

3.2.4.3.4 Application Requests Creating a File with a Security Descriptor

To create a file with a security descriptor, in addition to the parameters that are specified in section 3.2.4.3, the application MUST provide a buffer with a **SECURITY_DESCRIPTOR** in the format as specified in [MS-DTYP] section **2.4.6**. The client MUST construct a create context using the syntax specified for SMB2_CREATE_SD_BUFFER in section 2.2.13.2.2, and append it to any other create contexts being issued and pass it in for create processing.

3.2.4.3.5 Application Requests Creating a File Opened for Durable Operation

To request durable operation on a file being opened or created, in addition to the parameters that are specified in section 3.2.4.3, the application MUST provide a Boolean indicating whether durability is requested. If the application is requesting durability, the client MUST construct a create context using the syntax specified in section 2.2.13.2.3. The client MUST append it to any other create contexts being issued and pass it in for create processing. If the application is not requesting durability, the client MUST follow the normal processing, as specified in section 3.2.4.3.

3.2.4.3.6 Application Requests Opening a Previous Version of a File

To open a previous version of a file, in addition to the parameters that are specified in section 3.2.4.3, the application MUST provide a time stamp for the version to be opened, in **FILETIME** format as specified in [MS-DTYP] section 2.3.1. The client MUST construct a create context following the syntax as specified in section 2.2.13.2.7 using this time stamp. The client MUST append it to any other create contexts being issued and pass it in for create processing.

3.2.4.3.7 Application Requests Creating a File with a Specific Allocation Size

To create a file with a specific allocation size, in addition to the parameters specified in section 3.2.4.3, the application MUST provide an allocation size in LARGE_INTEGER format. The client MUST construct a create context following the syntax that is specified in section 2.2.13.2.6 and using this

allocation size. The client MUST append it to any other create contexts being issued and pass it in for create processing.

3.2.4.3.8 Re-establishing a Durable Open

To re-establish an open that existed on a connection that was previously disconnected, the client MUST locate the Open structure in the GlobalOpenTable. The client MUST attempt to connect to the target share, as specified in section 3.2.4.2. If this attempt fails, the client MUST fail the re-establishment attempt. If this attempt succeeds, the client MUST construct an SMB2 CREATE Request according to the syntax specified in section 2.2.13. The SMB2 header MUST be initialized as follows:

- The **Command** field is set to SMB2 CREATE.
- A sequence number is allocated from **Connection.SequenceWindow**. If no sequence number is available, it MUST wait for one to become available. The client sets **MessageId** to the sequence number that is received from **Connection.SequenceWindow**.
- The **SessionId** field is set to **TreeConnect.Session.SessionId**.
- The **TreeId** field is set to TreeConnect.TreeConnectId.
- The **ProcessId** field is set to a 4-byte identifier for the process that issued the original create request.

The SMB2 CREATE request MUST be initialized as follows:

- The **SecurityFlags** field is set to 0.
- The **RequestedOplockLevel** field is set to 0.
- The **ImpersonationLevel** field is set to 0.
- The client sets the **DesiredAccess** field to 0.
- The client sets the **FileAttributes** field to 0.
- The client sets the **ShareAccess** field to 0.
- The client sets the **CreateDisposition** field to 0.
- The client sets the **CreateOptions** field to 0.
- The client copies the relative path into **Buffer** and sets **NameLength** to the length, in bytes, of the relative path, and **NameOffset** to the offset, in bytes, to the relative path from the beginning of the SMB2 header.
- An SMB2_CREATE_DURABLE_HANDLE_RECONNECT create context is constructed according to the syntax specified in section 2.2.13.2.4. The data value is set to **Open.FileId**, and the create context is appended to the create request.

This request MUST be sent to the server.

3.2.4.4 Application Requests Closing a File or Named Pipe

The application MUST provide:

- The local identifier of the open to a file or named pipe that is being closed.

- A Boolean that, if set, specifies that it wants the attributes of the file after the close is executed.

The client MUST locate this open in the GlobalOpenTable. If no open is found, the client MUST fail the application request.

If the open is found, the **Open.Connection** is NULL, and the **Open.Durable** is TRUE, the client MAY attempt to reconnect to this open, as specified in section 3.2.4.3.8. If the reconnect succeeds, the close MUST be retried. If it fails, the error code MUST be returned to the application.<52>

If the open is found, the **Open.Connection** is NULL, and the **Open.Durable** is FALSE, the client MUST fail the close operation.

If the open is found and the **Open.Connection** is not NULL, the client MUST initialize an SMB2 CLOSE Request by following the syntax specified in section 2.2.15. The SMB2 header MUST be initialized as follows:

- The **Command** field is set to SMB2 CLOSE.
- A sequence number is allocated from **Connection.SequenceWindow**. If no sequence number is available, it MUST wait for one to become available. The client sets **MessageId** to the sequence number received from **Connection.SequenceWindow**.
- The **SessionId** field is set to **Open.TreeConnect.Session.SessionId**.
- The **TreeId** field is set to **Open.TreeConnect.TreeConnectId**.
- The **ProcessId** field is set to a 4-byte identifier for the process that issued the close request.

The SMB2 CLOSE request MUST be initialized as follows:

- If the application wants to have the attributes of the file returned after close, the client sets SMB2_CLOSE_FLAG_POSTQUERY_ATTRIB to TRUE in the Flags field.
- The **FileId** field is set to **Open.FileId**.

This request MUST be sent to the server.

3.2.4.5 Application Requests Reading from a File or Named Pipe

The application MUST provide:

- The local identifier of the open of a file or named pipe.
- The offset from which to read data.
- The number of bytes to read.
- The minimum number of bytes it would like to be read (optional).

The client MUST locate this open in the GlobalOpenTable. If no open is found, the client MUST fail the application request.

If the open is found, the **Open.Connection** is NULL, and the **Open.Durable** is TRUE, the client MAY attempt to reconnect to this open, as specified in section 3.2.4.3.8. If the reconnect succeeds, the read MUST be retried. If it fails, the error code MUST be returned to the application.<53>

If the open is found, the **Open.Connection** is NULL, and the **Open.Durable** is **FALSE**, the client MUST fail the read operation.

If the open is found, and the **Open.Connection** is not NULL, the client initializes an SMB2 READ Request following the syntax specified in section 2.2.19. The SMB2 header MUST be initialized as follows:

- The **Command** field is set to SMB2 READ.
- A sequence number is allocated from the **Connection.SequenceWindow**. If no sequence number is available, it MUST wait for one to become available. The client sets **MessageId** to the sequence number received from **Connection.SequenceWindow**.
- The **SessionId** field is set to **Open.TreeConnect.Session.SessionId**.
- The **TreeId** field is set to **Open.TreeConnect.TreeConnectId**.
- The **ProcessId** field is set to a 4-byte identifier for the process that issued the read request.

The SMB2 READ Request MUST be initialized as follows:

- The **Length** field is set to the number of bytes the application requested to read.
- The **Offset** field is set to the offset within the file, in bytes, at which the application requested the read to start.
- The **MinimumCount** field is set to the value that is provided by the application. If no value is provided by the application, the client MUST set this field to 0.
- The **FileId** field is set to **Open.FileId**.
- The **Padding** field SHOULD be set to 0x50, which is the default padding of an SMB2 READ Response.

If the number of bytes to read exceeds **Connection.MaxReadSize**, the client MUST split the read up into separate read operations no larger than **Connection.MaxReadSize**. The client MAY send these separate reads in any order.<54>

The request MUST be sent to the server.

3.2.4.6 Application Requests Writing to a File or Named Pipe

The application MUST provide:

- The local identifier of the open of a file or named pipe.
- The offset, in bytes, from where data should be written.
- The number of bytes to write.
- A buffer containing the bytes to be written.

The client MUST locate this open in the GlobalOpenTable. If no open is found, the client MUST fail the application request.

If the open is found, the **Open.Connection** is NULL, and the **Open.Durable** is TRUE, the client MAY attempt to reconnect to this open as specified in section 3.2.4.3.8. If the reconnect succeeds, the write MUST be retried. If it fails, the error code MUST be returned to the application.<55>

If the open is found, the **Open.Connection** is NULL and the **Open.Durable** is FALSE, the client MUST fail the write operation.

If the open is found and the **Open.Connection** is not NULL, the client initializes an SMB2 WRITE Request, following the syntax specified in section 2.2.21. The SMB2 header MUST be initialized as follows:

- The **Command** field is set to SMB2 WRITE.
- A sequence number is allocated from the **Connection.SequenceWindow**. If no sequence number is available, it MUST wait for one to become available. The client sets **MessageId** to the sequence number received from the **Connection.SequenceWindow**.
- The **SessionId** field is set to **Open.TreeConnect.Session.SessionId**.
- The **TreeId** field is set to **Open.TreeConnect.TreeConnectId**.
- The **ProcessId** field is set to a 4-byte identifier for the process that issued the write request.

The SMB2 WRITE Request MUST be initialized as follows:

- The **Length** field is set to the number of bytes the application requested to write.
- The **Offset** field is set to the offset within the file, in bytes, at which the application requested the write to start.
- The **FileId** field is set to **Open.FileId**.
- The **DataOffset** field is set to the offset from the beginning of the SMB2 header to the data being written. This value SHOULD be 0x70, which is the default offset for write requests.
- The data being written is copied into the request at DataOffset bytes from the beginning of the SMB2 header.
- The client MUST<56> fill the bytes, if any, between the beginning of the **Buffer** field and the beginning of the data (at **DataOffset**) with zeros.

If the number of bytes to write exceeds the **Connection.MaxWriteSize**, the client MUST split the write into separate write operations no larger than the **Connection.MaxWriteSize**. The client MAY send these separate writes in any order.<57>

The request MUST be sent to the server.

3.2.4.7 Application Requests Querying File Attributes

The application MUST provide:

- The local identifier of the open of a file or named pipe.
- The maximum output buffer it will accept.
- The InformationClass of the attributes being queried, as specified in [MS-FSCC] section 2.4.
- If the information being queried is **FileFullEaInformation**, the application also provides the starting index, in bytes, of the extended attribute information to be returned.

The client MUST locate this open in GlobalOpenTable. If no open is found, the client MUST fail the application request.

If the open is found, the **Open.Connection** is NULL, and the **Open.Durable** is **TRUE**, the client MAY attempt to reconnect to this open, as specified in section 3.2.4.3.8. If the reconnect succeeds, the query MUST be retried. If it fails, the error code MUST be returned to the application.<58>

If the open is found, the **Open.Connection** is NULL, and the **Open.Durable** is FALSE, the client MUST fail the query operation.

If the open is found and the **Open.Connection** is not NULL, the client initializes an SMB2 QUERY_INFO Request following the syntax specified in section 2.2.37. The SMB2 header MUST be initialized as follows:

- The **Command** field is set to SMB2 QUERY_INFO.
- A sequence number is allocated from **Connection.SequenceWindow**. If no sequence number is available, it MUST wait for one to become available. The client sets **MessageId** to the sequence number received from **Connection.SequenceWindow**.
- The **SessionId** field is set to **Open.TreeConnect.Session.SessionId**.
- The **TreeId** field is set to **Open.TreeConnect.TreeConnectId**.
- The **ProcessId** field is set to a 4-byte identifier for the process that issued the query info request.

The SMB2 QUERY_INFO Request MUST be initialized as follows:

- The **InfoType** field is set to SMB2_0_INFO_FILE.
- The **FileInfoClass** field is set to the **InformationLevel** received from the application.
- The **OutputBufferLength** field is set to the maximum output buffer the calling application will accept.
- The **InputBufferOffset** field is set to 0.
- The **InputBufferLength** field is set to 0.
- If the query is for **FileFullEaInformation**, the **AdditionalInformation** is set to the extended attribute index that is received from the calling application. Otherwise, it is set to 0.
- The **FileId** field is set to **Open.FileId**.

The request MUST be sent to the server.

3.2.4.8 Application Requests Applying File Attributes

The application MUST provide:

- The local identifier of the open of a file or named pipe.
- The InformationClass of the information being applied to the file or pipe, as specified in [MS-FSCC] section 2.4.
- A buffer containing the information being applied.

The client MUST locate this open in the GlobalOpenTable. If no open is found, the client MUST fail the application request.

If the open is found, the **Open.Connection** is NULL, and the **Open.Durable** is TRUE, the client MAY attempt to reconnect to this open, as specified in section 3.2.4.3.8. If the reconnect succeeds, the set MUST be retried. If it fails, the error code MUST be returned to the application.<59>

If the open is found, the **Open.Connection** is NULL, and the **Open.Durable** is FALSE, the client MUST fail the set operation.

If the open is found and the **Open.Connection** is not NULL, the client initializes an SMB2 SET_INFO Request following the syntax specified in section 2.2.37. The SMB2 header MUST be initialized as follows:

- The **Command** field is set to SMB2 SET_INFO.
- A sequence number is allocated from **Connection.SequenceWindow**. If no sequence number is available, it MUST wait for one to become available. The client sets the **MessageId** to the sequence number that is received from the **Connection.SequenceWindow**.
- The **SessionId** field is set to **Open.TreeConnect.Session.SessionId**.
- The **TreeId** field is set to **Open.TreeConnect.TreeConnectId**.
- The **ProcessId** field is set to a 4-byte identifier for the process that issued the set information request.

The SMB2 SET_INFO request MUST be initialized as follows:

- The **InfoType** field is set to SMB2_0_INFO_FILE.
- The **FileInfoClass** field is set to the InformationClass provided by the application.
- The buffer provided by the client is copied into Buffer[].
- The **BufferOffset** field is set to the offset, in bytes, from the beginning of the SMB2 header to Buffer[].
- The **BufferLength** field is set to the length, in bytes, of the buffer that is provided by the application.
- The **AdditionalInformation** is set to 0.
- The **FileId** field is set to **Open.FileId**.

The request MUST be sent to the server.

3.2.4.9 Application Requests Querying File System Attributes

The application MUST provide:

- The local identifier of the open of a file on the file system whose attributes are being queried.
- The maximum output buffer it will accept.
- The InformationClass of the file system attributes being queried, as specified in [MS-FSCC] section 2.5.

The client MUST locate this open in GlobalOpenTable. If no open is found, the client MUST fail the application request.

If the open is found, the **Open.Connection** is NULL, and the **Open.Durable** is **TRUE**, the client MAY attempt to reconnect to this open, as specified in section 3.2.4.3.8. If the reconnect succeeds, the query MUST be retried. If it fails, the error code MUST be returned to the application.<60>

If the open is found, the **Open.Connection** is NULL, and the **Open.Durable** is FALSE, the client MUST fail the query operation.

If the open is found and the **Open.Connection** is not NULL, the client initializes an SMB2 QUERY_INFO Request following the syntax specified in section 2.2.37. The SMB2 header MUST be initialized as follows:

- The **Command** field is set to SMB2 QUERY_INFO.
- A sequence number is allocated from **Connection.SequenceWindow**. If no sequence number is available, it MUST wait for one to become available. The client sets **MessageId** to the sequence number that is received from **Connection.SequenceWindow**.
- The **SessionId** field is set to **Open.TreeConnect.Session.SessionId**.
- The **TreeId** field is set to **Open.TreeConnect.TreeConnectId**.
- The **ProcessId** field is set to a 4-byte identifier for the process that issued the query request.

The SMB2 QUERY_INFO Request MUST be initialized as follows:

- The **InfoType** field is set to SMB2_0_INFO_FILESYSTEM.
- The **FileInfoClass** field is set to the **InformationLevel** that is received from the application.
- The **OutputBufferLength** field is set to the maximum output buffer that the calling application will accept.
- The **InputBufferOffset** field is set to 0.
- The **InputBufferLength** field is set to 0.
- The **AdditionalInformation** is set to 0.
- The **FileId** field is set to **Open.FileId**.

The request MUST be sent to the server.

3.2.4.10 Application Requests Applying File System Attributes

The application MUST provide:

- The local identifier of the open of a file on the file system whose attributes are being changed.
- The InformationClass of the file system attributes being queried, as specified in [MS-FSCC] section 2.5.
- A buffer that contains the attributes being applied.

The client MUST locate this open in the GlobalOpenTable. If no open is found, the client MUST fail the application request.

If the open is found, the **Open.Connection** is NULL, and the **Open.Durable** is TRUE, the client MAY attempt to reconnect to this open, as specified in section 3.2.4.3.8. If the reconnect succeeds, the set MUST be retried. If it fails, the error code MUST be returned to the application.<61>

If the open is found, the **Open.Connection** is NULL, and the **Open.Durable** is **FALSE**, the client MUST fail the set operation.

If the open is found and the **Open.Connection** is not NULL, the client initializes an SMB2 SET_INFO Request following the syntax specified in section 2.2.37. The SMB2 header MUST be initialized as follows:

- The **Command** field is set to SMB2 SET_INFO.
- A sequence number is allocated from **Connection.SequenceWindow**. If no sequence number is available, it MUST wait for one to become available. The client sets the **MessageId** to the sequence number that is received from **Connection.SequenceWindow**.
- The **SessionId** field is set to **Open.TreeConnect.Session.SessionId**.
- The **TreeId** field is set to **Open.TreeConnect.TreeConnectId**.
- The **ProcessId** field is set to a 4-byte identifier for the process that issued the set information request.

The SMB2 SET_INFO request MUST be initialized as follows:

- The **InfoType** field is set to SMB2_0_INFO_FILESYSTEM.
- The **FileInfoClass** field is set to the InformationClass that is provided by the application.
- The buffer provided by the application is copied into Buffer[].
- The **BufferOffset** field is set to the offset, in bytes, from the beginning of the SMB2 header to **Buffer**[].
- The **BufferLength** field is set to the length, in bytes, of the buffer that is provided by the application.
- The **AdditionalInformation** is set to 0.
- The **FileId** field is set to **Open.FileId**.

The request MUST be sent to the server.

3.2.4.11 Application Requests Querying File Security

The application MUST provide:

- The local identifier of the open of a file or named pipe.
- The maximum output buffer it will accept.
- The security attributes it is querying for the file, as specified in the AdditionalInformation description of section 2.2.37.

The client MUST locate this open in the GlobalOpenTable. If no open is found, the client MUST fail the application request.

If the open is found, the **Open.Connection** is NULL, and the **Open.Durable** is TRUE, the client MAY attempt to reconnect to this open, as specified in section 3.2.4.3.8. If the reconnect succeeds, the query MUST be retried. If it fails, the error code MUST be returned to the application.<62>

If the open is found, the **Open.Connection** is NULL, and the **Open.Durable** is **FALSE**, the client MUST fail the query operation.

If the open is found and the **Open.Connection** is not NULL, the client initializes an SMB2 QUERY_INFO Request following the syntax specified in section 2.2.37. The SMB2 header MUST be initialized as follows:

- The **Command** field is set to SMB2 QUERY_INFO.
- A sequence number is allocated from **Connection.SequenceWindow**. If no sequence number is available, it MUST wait for one to become available. The client sets the **MessageId** to the sequence number that is received from the **Connection.SequenceWindow**.
- The **SessionId** field is set to **Open.TreeConnect.TreeConnectId**.
- The **TreeId** field is set to **Open.TreeConnect.TreeConnectId**.
- The **ProcessId** field is set to a 4-byte identifier for the process that issued the query request.

The SMB2 QUERY_INFO Request MUST be initialized as follows:

- The **InfoType** field is set to SMB2_0_INFO_SECURITY.
- The **FileInfoClass** field is set to 0.
- The **OutputBufferLength** field is set to the maximum output buffer that the calling application will accept.
- The **InputBufferOffset** field is set to 0.
- The **InputBufferLength** field is set to 0.
- The **AdditionalInformation** is set to the security attributes that are provided by the calling application.
- The **FileId** field is set to **Open.FileId**.

The request MUST be sent to the server.

3.2.4.12 Application Requests Applying File Security

The application MUST provide:

- The local identifier of the open of a file or named pipe.
- The security information being applied in security descriptor format, as specified in [MS-DTYP] section **2.4.6**.
- The security attributes it wants to set for the file, as specified in section 2.2.37.

The client MUST locate this open in the GlobalOpenTable. If no open is found, the client MUST fail the application request.

If the open is found, the **Open.Connection** is NULL, and the **Open.Durable** is **TRUE**, the client MAY attempt to reconnect to this open, as specified in section 3.2.4.3.8. If the reconnect succeeds, the set MUST be retried. If it fails, the error code MUST be returned to the application.<63>

If the open is found, the **Open.Connection** is NULL, and the **Open.Durable** is **FALSE**, the client MUST fail the set operation.

If the open is found and the **Open.Connection** is not NULL, the client initializes an SMB2 SET_INFO Request following the syntax specified in section 2.2.37. The SMB2 header MUST be initialized as follows:

- The **Command** field is set to SMB2 SET_INFO.
- A sequence number is allocated from **Connection.SequenceWindow**. If no sequence number is available, it MUST wait for one to become available. The client sets the **MessageId** to the sequence number that is received from the **Connection.SequenceWindow**.
- The **SessionId** field is set to **Open.TreeConnect.Session.SessionId**.
- The **TreeId** field is set to **Open.TreeConnect.TreeConnectId**.
- The **ProcessId** field is set to a 4-byte identifier for the process that issued the set information request.

The SMB2 SET_INFO Request (section 2.2.39) MUST be initialized as follows:

- The **InfoType** field is set to SMB2_0_INFO_SECURITY.
- The **FileInfoClass** field is set to 0.
- The security descriptor that is provided by the client is copied into **Buffer**[].
- The **BufferOffset** field is set to the offset, in bytes, from the beginning of the SMB2 header to **Buffer**[].
- The **BufferLength** field is set to the length, in bytes, of the security descriptor that is provided by the application.
- The **AdditionalInformation** is set to the security attributes that are provided by the calling application.
- The **FileId** field is set to **Open.FileId**.

The request MUST be sent to the server.

3.2.4.13 Application Requests Querying Quota Information

The application MUST provide:

- The local identifier of an open to a directory.
- A Boolean indicating whether the enumeration is being restarted.
- A Boolean indicating whether only a single entry is to be returned.
- The maximum output buffer it will accept.

- It also optionally can provide either a start SID to determine where the enumeration should begin, or a list of SIDs whose quota information is being requested.

The client MUST locate this open in the GlobalOpenTable. If no open is found, the client MUST fail the application request.

If the open is found, the **Open.Connection** is NULL, and the **Open.Durable** is **TRUE**, the client MAY attempt to reconnect to this open, as specified in section 3.2.4.3.8. If the reconnect succeeds, the query MUST be retried. If it fails, the error code MUST be returned to the application.<64>

If the open is found, the **Open.Connection** is NULL, and the **Open.Durable** is **FALSE**, the client MUST fail the query operation.

If the open is found and the **Open.Connection** is not NULL, the client initializes an SMB2 QUERY_INFO request following the syntax specified in section 2.2.37. The SMB2 header MUST be initialized as follows:

- The **Command** field is set to SMB2 QUERY_INFO.
- A sequence number is allocated from **Connection.SequenceWindow**. If no sequence number is available, it MUST wait for one to become available. The client sets the **MessageId** to the sequence number that is received from the **Connection.SequenceWindow**.
- The **SessionId** field is set to **Open.TreeConnect.Session.SessionId**.
- The **TreeId** field is set to **Open.TreeConnect.TreeConnectId**.
- The **ProcessId** field is set to a 4-byte identifier for the process that issued the query request.

The SMB2 QUERY_INFO Request MUST be initialized as follows:

- The **InfoType** field is set to SMB2_0_INFO_QUOTA.
- The **FileInfoClass** field is set to 0.
- The **OutputBufferLength** field is set to the maximum output buffer that the calling application will accept.
- The **AdditionalInformation** is set to 0.
- The **FileId** field is set to **Open.FileId**.
- An **SMB2_QUERY_QUOTA_INFO** structure is constructed and copied into the **Buffer**[], and initialized as follows:
  - If only a single entry is to be returned, the client sets ReturnSingleEntry to **TRUE**. Otherwise, it is set to FALSE.
  - If the application wants to restart the scan, the client sets RestartScan to **TRUE**. Otherwise, it is set to FALSE.
  - **SidListLength**, **StartSidOffset**, and **StartSidLength** are set based on the parameters received from the application as follows:
    - If a list of SIDs is provided, they are copied immediately after the **SMB2_QUERY_QUOTA_INFO base** structure,**SidListLength** is set to their length in bytes, **StartSidLength** is set to 0, and StartSidOffset is set to 0.

- If a start SID is provided, it is copied into the structure after the **SMB2_QUERY_QUOTA_INFO base** structure, **StartSidLength** is set to the length of the SID, **StartSidOffset** is set to the offset, in bytes, to the start SID from the beginning of the **SMB2_QUERY_QUOTA_INFO** structure, and **SidListLength** is set to 0.
- If neither a start SID nor a list of SIDs is provided by the application, then **SidListLength** is set to 0, **StartSidLength** is set to 0, and **StartSidOffset** is set to 0.

- The **InputBufferOffset** field is set to describe the length and offset from the beginning of the SMB2 header to the **SMB2_QUERY_QUOTA_INFO** structure.
- The **InputBufferLength** field is set to the size, in bytes, of the **SMB2_QUERY_QUOTA_INFO** structure, including any trailing buffers for either the starting SID or SID list.

The request MUST be sent to the server.

3.2.4.14 Application Requests Applying Quota Information

The application MUST provide:

- The local identifier of an open to a directory.
- The quota information being applied in the format specified in [MS-SMB] section 2.2.14.4.

The client MUST locate this open in the GlobalOpenTable. If no open is found, the client MUST fail the application request.

If the open is found, the **Open.Connection** is NULL, and the **Open.Durable** is **TRUE**, the client MAY attempt to reconnect to this open, as specified in section 3.2.4.3.8. If the reconnect succeeds, the set MUST be retried. If it fails, the error code MUST be returned to the application.<65>

If the open is found, the **Open.Connection** is NULL, and the **Open.Durable** is FALSE, the client MUST fail the set operation.

If the open is found and the **Open.Connection** is not NULL, the client initializes an SMB2 SET_INFO request, following the syntax specified in section 2.2.37. The SMB2 header MUST be initialized as follows:

- The **Command** field is set to SMB2 SET_INFO.
- A sequence number is allocated from the **Connection.SequenceWindow**. If no sequence number is available, it MUST wait for one to become available. The client sets the **MessageId** to the sequence number that is received from **Connection.SequenceWindow**.
- The **SessionId** field is set to **Open.TreeConnect.Session.SessionId**.
- The **TreeId** field is set to **Open.TreeConnect.TreeConnectId**.
- The **ProcessId** field is set to a 4-byte identifier for the process that issued the set information request.

The SMB2 SET_INFO request MUST be initialized as follows:

- The **InfoType** field is set to SMB2_0_INFO_QUOTA.
- The **FileInfoClass** field is set to 0.
- The quota information that is provided by the client is copied into the **Buffer**[].

- The **BufferOffset** field is set to the offset, in bytes, from the beginning of the SMB2 header to the Buffer[].
- The **BufferLength** field is set to the length, in bytes, of the quota information that is provided by the application.
- The **AdditionalInformation** is set to 0.
- The **FileId** field is set to **Open.FileId**.

The request MUST be sent to the server.

3.2.4.15 Application Requests Flushing Cached Data

The application MUST provide:

- The local identifier of the open of a file or named pipe for which it wants to flush cached data.

The client MUST locate this open in the GlobalOpenTable. If no open is found, the client MUST fail the application request.

If the open is found, the **Open.Connection** is NULL, and the **Open.Durable** is TRUE, the client MAY attempt to reconnect to this open, as specified in section 3.2.4.3.8. If the reconnect succeeds, the flush MUST be retried. If it fails, the error code MUST be returned to the application.<66>

If the open is found, the **Open.Connection** is NULL, and the **Open.Durable** is **FALSE**, the client MUST fail the flush operation.

If the open is found and the **Open.Connection** is not NULL, the client initializes an SMB2 FLUSH request by following the syntax specified in section 2.2.17. The SMB2 header MUST be initialized as follows:

- The **Command** field is set to SMB2 FLUSH.
- A sequence number is allocated from **Connection.SequenceWindow**. If no sequence number is available, it MUST wait for one to become available. The client sets the **MessageId** to the sequence number that is received from the **Connection.SequenceWindow**.
- The **SessionId** field is set to **Open.TreeConnect.Session.SessionId**.
- The **TreeId** field is set to **Open.TreeConnect.TreeConnectId**.
- The **ProcessId** field is set to a 4-byte identifier for the process that issued the flush request.

The SMB2 FLUSH request MUST be initialized as follows:

- The **FileId** field is set to **Open.FileId**.

The request MUST be sent to the server.

3.2.4.16 Application Requests Enumerating a Directory

The application MUST provide:

- The local identifier of the open to a directory.
- The InformationClass of the file information being queried, as specified in [MS-FSCC] section 2.4.

- The maximum buffer size it will accept in response.
- A Boolean indicating whether the enumeration should be restarted.
- A Boolean indicating whether only a single entry should be returned.
- A Boolean indicating whether the file specifier has been changed if the enumeration is being restarted.
- A 4-byte index number to resume the enumeration from if the destination file system supports it (optional).
- A file specifier string for the enumeration.

The client MUST locate this open in the GlobalOpenTable. If no open is found, the client MUST fail the application request.

If the open is found, the **Open.Connection** is NULL, and the Open.Durable is TRUE, the client MAY attempt to reconnect to this open, as specified in section 3.2.4.3.8. If the reconnect succeeds, the enumeration MUST be retried. If it fails, the error code MUST be returned to the application.<67>

If the open is found, the **Open.Connection** is NULL, and the Open.Durable is FALSE, the client MUST fail the enumeration operation.

If the open is found and the **Open.Connection** is not NULL, the client initializes an SMB2 QUERY_DIRECTORY request, following the syntax specified in section 2.2.37. The SMB2 header MUST be initialized as follows:

- The **Command** field is set to SMB2 QUERY_DIRECTORY.
- A sequence number is allocated from Connection.SequenceWindow. If no sequence number is available, it MUST wait for one to become available. The client sets MessageId to the sequence number received from Connection.SequenceWindow.
- The SessionId field is set to Open.TreeConnect.Session.SessionId.
- The **TreeId** field is set to Open.TreeConnect.TreeConnectId.
- The ProcessId field is set to a 4-byte identifier for the process that issued the enumerate request.

The SMB2 QUERY_DIRECTORY request MUST be initialized as follows:

- The FileInformationClass field is set to the InformationClass that is received from the application.
- The OutputBufferLength field is set to the maximum output buffer that the calling application will accept.
- If a file specifier string is provided, the client copies it into the Buffer[] and sets the FileNameOffset to the offset, in bytes, from the beginning of the SMB2 header to the start of the Buffer[]; and the FileNameLength to the length, in bytes, of the file specifier string. Otherwise, it sets FileNameOffset and FileNameLength to 0.
- If a file index was provided by the application, the client sets the value in the FileIndex field and sets SMB2_INDEX_SPECIFIED to TRUE in the Flags field.
- The FileId field is set to Open.FileId.

- If the application requested that the enumeration be restarted, the client sets SMB2_RESTART_SCANS to TRUE in the Flags field.
- If the application requested that the enumeration be restarted and indicated that the file specifier has changed, the client sets SMB2_REOPEN to TRUE in the Flags field. If there are other operations outstanding on this open (change notifications, other enumerations, attribute queries, and so on), the client MUST fail this request.
- If the application requested that only a single entry be returned, the client sets SMB2_RETURN_SINGLE_ENTRY to TRUE in the Flags field.

The request MUST be sent to the server.

3.2.4.16.1 Application Requests Continuing a Directory Enumeration

If an application wants to continue an enumeration for which only partial results were previously returned, it does so by executing a request, as specified in section 3.2.4.16, but makes sure it does not request restarting the enumeration. Doing so allows it to continue a previous enumeration.

3.2.4.17 Application Requests Change Notifications for a Directory

The application MUST provide:

- The local identifier of the open of a directory.
- The maximum output buffer it will accept.
- A Boolean indicating whether the directory should be monitored recursively.
- The completion filter following the syntax specified in section 2.2.35, denoting which changes the application would like to be notified of.

If the application wants to be notified when changes occur and does not need to see the actual changes, the maximum output buffer MUST be set to 0.

The client MUST locate this open in the GlobalOpenTable. If no open is found, the client MUST fail the application request.

If the open is found, the Open.Connection is NULL, and the Open.Durable is TRUE, the client MAY attempt to reconnect to this open, as specified in section 3.2.4.3.8. If the reconnect succeeds, the change notify MUST be retried. If it fails, the error code MUST be returned to the application.<68>

If the open is found, the Open.Connection is NULL, and the Open.Durable is FALSE, the client MUST fail the change notify operation.

If the open is found and the Open.Connection is not NULL, the client initializes an SMB2 CHANGE_NOTIFY request, following the syntax specified in section 2.2.37. The SMB2 header MUST be initialized as follows:

- The Command field is set to SMB2 CHANGE_NOTIFY.
- A sequence number is allocated from the Connection.SequenceWindow. If no sequence number is available, it MUST wait for one to become available. The client sets the MessageId to the sequence number that is received from the Connection.SequenceWindow.
- The SessionId field is set to Open.TreeConnect.Session.SessionId.

- The **TreeId** field is set to Open.TreeConnect.TreeConnectId.
- The ProcessId field is set to a 4-byte identifier for the process that issued the change notify request.

The SMB2 CHANGE_NOTIFY request MUST be initialized as follows:

- The CompletionFilter field is set to the completion filter that is provided by the calling application.
- The OutputBufferLength field is set to the maximum output buffer that the calling application will accept.
- The FileId field is set to Open.FileId.
- If the application requested that the directory be monitored recursively, the client sets SMB2_WATCH_TREE to TRUE in the Flags field.

The request MUST be sent to the server.

3.2.4.18 Application Requests Locking of an Array of Byte Ranges

The application MUST provide:

- The local identifier of the open of a file or named pipe.
- An array of byte ranges to lock. For each range, the application provides:
  - A starting offset, in bytes.
  - A length, in bytes.
  - Whether the range is to be locked exclusively or shared.
  - Whether the lock request is to wait until the lock can be acquired to return, or whether it is to fail immediately if the range is locked by another open.

The client MUST locate this open in the GlobalOpenTable. If no open is found, the client MUST fail the application request.

If the open is found, the Open.Connection is NULL, and the Open.Durable is TRUE, the client MAY attempt to reconnect to this open as specified in section 3.2.4.3.8. If the reconnect succeeds, the lock MUST be retried. If it fails, the error code MUST be returned to the application.<69>

If the open is found, the Open.Connection is NULL, and the Open.Durable is FALSE, the client MUST fail the read operation.

If the open is found and the Open.Connection is not NULL, the client initializes an SMB2 LOCK request following the syntax specified in section 2.2.26. The SMB2 header MUST be initialized as follows:

- The Command field is set to SMB2 LOCK.
- A sequence number is allocated from Connection.SequenceWindow. If no sequence number is available, it MUST wait for one to become available. The client sets the MessageId to the sequence number that is received from the Connection.SequenceWindow.
- The SessionId field is set to Open.TreeConnect.Session.SessionId.

- The **TreeId** field is set to Open.TreeConnect.TreeConnectId.
- The ProcessId field is set to a 4-byte identifier for the process that issued the lock request.

The SMB2 LOCK request MUST be initialized as follows:

- The FileId field is set to Open.FileId.
- The LockCount field is set to the number of byte ranges being locked.
- For each range being locked, the client creates an SMB2_LOCK structure and places it in the Locks[] array of the request, setting the following values:
  - The offset is set to the offset of the range being locked.
  - The length is set to the length of the range to be locked.
  - If the lock is to be acquired shared, the client sets SMB2_LOCKFLAG_SHARED_LOCK to TRUE in the Flags field.
  - If the lock is to be acquired exclusively, the client sets SMB2_LOCKFLAG_EXCLUSIVE_LOCK to TRUE in the Flags field.
  - If the lock is to fail immediately if the range is already locked, the client sets SMB2_LOCKFLAG_FAIL_IMMEDIATELY to TRUE in the Flags field. If the Locks[] array has more than one element, the client MUST set SMB2_LOCKFLAG_FAIL_IMMEDIATELY.

The request MUST be sent to the server.

3.2.4.19 Application Requests an IO Control Code Operation

3.2.4.19.1 Application Requests Enumeration of Previous Versions

The application MUST provide:

- The local identifier of the open of a file on a volume for which the application wants the previous version time stamps.
- The maximum output buffer size that it will accept.

The client MUST locate the open in the GlobalOpenTable. If no open is found, the client MUST fail the application request.

If the open is found, the Open.Connection is NULL, and the Open.Durable is TRUE, the client MAY attempt to reconnect to this open, as specified in section 3.2.4.3.8. If the reconnect succeeds, this FSCTL MUST be retried. If it fails, the error code MUST be returned to the application.<70>

If the open is found, the Open.Connection is NULL, and the Open.Durable is FALSE, the client MUST fail this FSCTL operation.

If the open is found and the Open.Connection is not NULL, the client initializes an SMB2 IOCTL request following the syntax specified in section 2.2.31. The SMB2 header MUST be initialized as follows:

- The Command field is set to SMB2 IOCTL.

- A sequence number is allocated from the Connection.SequenceWindow. If no sequence number is available, it MUST wait for one to become available. The client sets the MessageId to the sequence number that is received from the Connection.SequenceWindow.
- The SessionId field is set to Open.TreeConnect.Session.SessionId.
- The **TreeId** field is set to Open.TreeConnect.TreeConnectId.
- The ProcessId field is set to a 4-byte identifier for the process that issued the IOCTL request.

The SMB2 IOCTL request MUST be initialized as follows:

- The **CtlCode** field is set to FSCTL_SRV_ENUMERATE_SNAPSHOTS.
- The FileId field is set to Open.FileId.
- The InputOffset field is set to the offset to the Buffer[], in bytes, from the beginning of the SMB2 header.
- The InputCount field is set to 0.
- The OutputOffset field SHOULD be set to zero. <71>
- The OutputCount field is set to 0.
- The MaxInputResponse field is set to 0.
- The MaxOutputResponse field is set to the maximum output buffer size that the application will accept.
- SMB2_0_IOCTL_IS_FSCTL is set to TRUE in the Flags field.

The request MUST be sent to the server.

3.2.4.19.2 Application Requests a Server-Side Data Copy

Requesting a server-side data copy occurs in several steps. These are outlined in the following diagram:

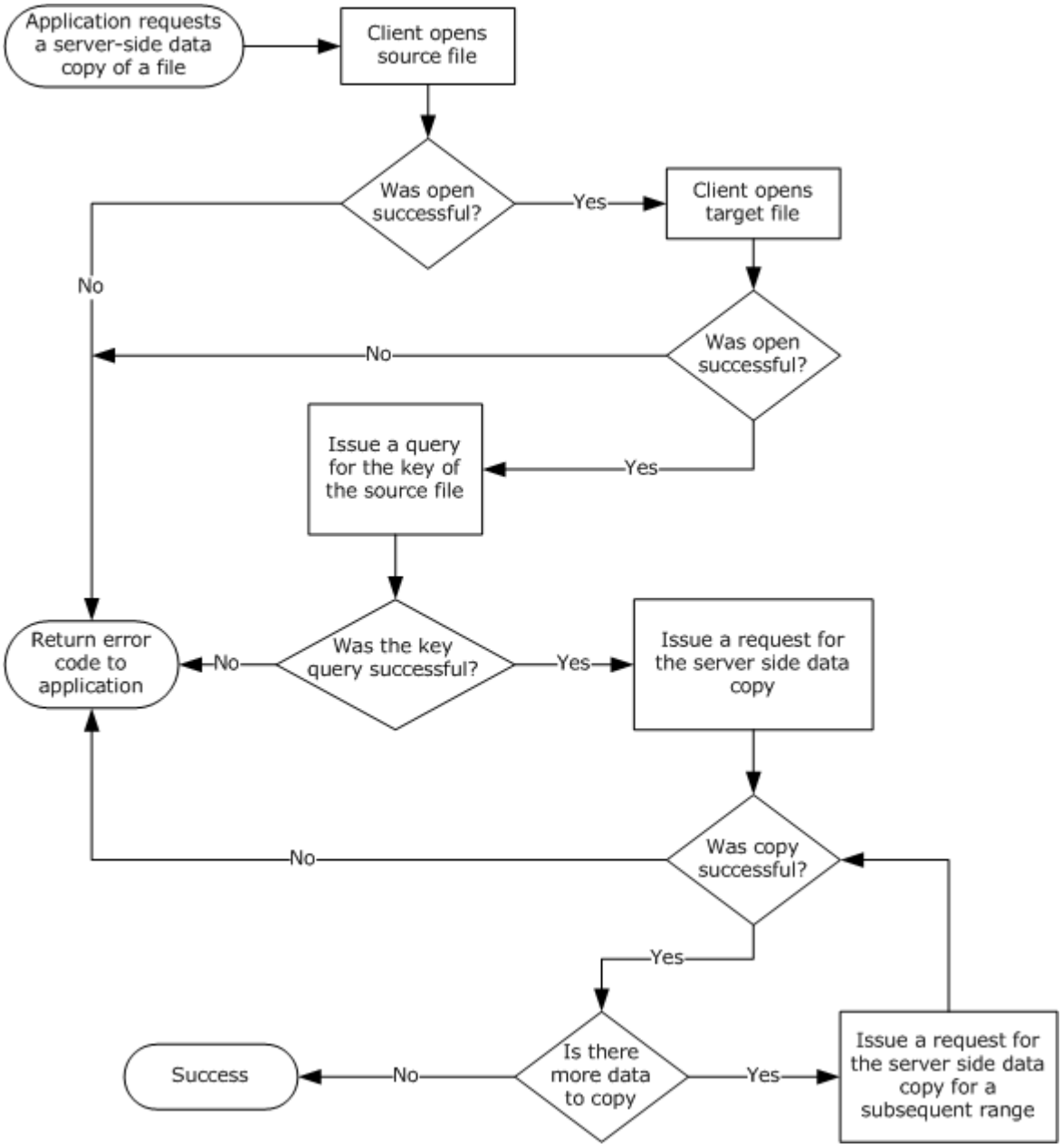


Figure 2: Application requesting a server-side data copy

The method for requesting the key to the source file and for requesting the server-side copy of data ranges is outlined in the following sections.

3.2.4.19.2.1 Application Requests a Source File Key

The application MUST provide:

- The local identifier of the open of a file for which the application wants a key to use in server-side data operations.

The client MUST locate the open in the GlobalOpenTable. If no open is found, the client MUST fail the application request.

If the open is found, the Open.Connection is NULL, and the Open.Durable is TRUE, the client MAY attempt to reconnect to this open, as specified in section 3.2.4.3.8. If the reconnect succeeds, this FSCTL MUST be retried. If it fails, the error code MUST be returned to the application.<72>

If the open is found, the Open.Connection is NULL, and the Open.Durable is FALSE, the client MUST fail this FSCTL operation.

If the open is found and the Open.Connection is not NULL, the client initializes an SMB2 IOCTL request following the syntax specified in section 2.2.31. The SMB2 header MUST be initialized as follows:

- The Command field is set to SMB2 IOCTL.
- A sequence number is allocated from Connection.SequenceWindow. If no sequence number is available, it MUST wait for one to become available. The client sets MessageId to the sequence number received from Connection.SequenceWindow.
- The SessionId field is set to Open.TreeConnect.Session.SessionId.
- The **TreeId** field is set to Open.TreeConnect.TreeConnectId.
- The ProcessId field is set to a 4-byte identifier for the process that issued the IOCTL request.

The SMB2 IOCTL request MUST be initialized as follows:

- The **CtlCode** field is set to the FSCTL_SRV_REQUEST_RESUME_KEY.
- The FileId field is set to Open.FileId.
- The InputOffset field is set to the offset to the Buffer[], in bytes, from the beginning of the SMB2 header.
- The InputCount field is set to 0.
- The OutputOffset field is set to InputOffset + InputCount, rounded up to a multiple of 8 bytes.
- The OutputCount field is set to 0.
- The MaxInputResponse field is set to 0.
- The MaxOutputResponse field is set to 32.
- SMB2_0_IOCTL_IS_FSCTL is set to TRUE in the Flags field.

The request MUST be sent to the server.

3.2.4.19.2.2 Application Requests a Server Side Data Copy

The application MUST provide:

- The local identifier of the open of the destination file.
- The key for the source file queried, as specified in the previous section "Application Requests a Source File Key."

- An array of ranges to copy. Each item in the array MUST contain the source offset, the destination offset, and the number of bytes to copy.

The client MUST locate the open in the GlobalOpenTable. If no open is found, the client MUST fail the application request.

If the open is found, the Open.Connection is NULL, and the Open.Durable is TRUE, the client MAY attempt to reconnect to this open, as specified in section 3.2.4.3.8. If the reconnect succeeds, this FSCTL MUST be retried. If it fails, the error code MUST be returned to the application.<73>

If the open is found, the Open.Connection is NULL, and the Open.Durable is FALSE, the client MUST fail this FSCTL operation.

If the open is found and the Open.Connection is not NULL, the client initializes an SMB2 IOCTL request following the syntax specified in section 2.2.31. The SMB2 header MUST be initialized as follows:

- The Command field is set to SMB2 IOCTL.
- A sequence number is allocated from Connection.SequenceWindow. If no sequence number is available, it MUST wait for one to become available. The client sets MessageId to the sequence number received from the Connection.SequenceWindow.
- The SessionId field is set to Open.TreeConnect.Session.SessionId.
- The **TreeId** field is set to Open.TreeConnect.TreeConnectId.
- The ProcessId field is set to a 4-byte identifier for the process that issued the IOCTL request.

The SMB2 IOCTL request MUST be initialized as follows:

- The **CtlCode** field is set to FSCTL_SRV_COPYCHUNK.
- The FileId field is set to Open.FileId.
- The InputOffset field is set to the offset to the Buffer[], in bytes, from the beginning of the SMB2 header.
- The InputCount is set to the size, in bytes, of the SRV_COPYCHUNK_COPY structure that is constructed following the syntax specified in section 2.2.31.1.1 with the client input parameters as follows:
  - The client sets the SourceKey field to the key of the source file.
  - For each range to be copied, the client initializes a SRV_COPYCHUNK structure following the syntax specified in section 2.2.31.1.1 using the provided source offset, destination offset, and length, in bytes.
  - The ChunkCount is set to the number of chunks being sent.
- The SRV_COPYCHUNK_COPY structure is copied into the request at InputOffset bytes from the beginning of the SMB2 header.
- The OutputOffset field SHOULD be set to zero. <74>
- The OutputCount field is set to 0.
- The MaxInputResponse field is set to 0.

- The MaxOutputResponse field is set to the size of a SRV_COPYCHUNK_RESPONSE structure, as specified in section 2.2.32.1.
- SMB2_0_IOCTL_IS_FSCTL is set to TRUE in the Flags field.

The request MUST be sent to the server.

3.2.4.19.3 Application Requests DFS Referral Information

The application MUST provide:

- The name of the server.
- The name of the share.
- The security principal that should be used to query the Distributed File System (DFS) referral information.
- The maximum output buffer response size, in bytes.
- The path for which referral information is to be queried.
- The maximum DFS referral level that the application will accept.

The client MUST locate the connection in the ConnectionTable. If no connection is found, it MUST fail the request.

Next, the client MUST locate the session for this security principal in the Connection.SessionTable. If no session is found, the client MUST fail the request.

The client MUST then locate the tree connect for this share in the Session.TreeConnectTable. If no tree connect is found, it MUST fail the request.

The client initializes an SMB2 IOCTL request following the syntax specified in section 2.2.31. The SMB2 header MUST be initialized as follows:

- The **Command** field is set to SMB2 IOCTL.
- A sequence number is allocated from Connection.SequenceWindow. If no sequence number is available, it MUST wait for one to become available. The client sets MessageId to the sequence number received from Connection.SequenceWindow.
- The **SessionId** field is set to Open.TreeConnect.Session.SessionId.
- The **TreeId** field is set to Open.TreeConnect.TreeConnectId.
- The **ProcessId** field is set to a 4-byte identifier for the process that issued the IOCTL request.

The SMB2 IOCTL request MUST be initialized as follows:

- The **CtlCode** field is set to the operation code to FSCTL_DFS_GET_REFERRALS.
- The **FileId** field is set to { 0xFFFFFFFFFFFFFFFF, 0xFFFFFFFFFFFFFFFF }.
- The **InputOffset** field is set to the offset to the Buffer[], in bytes, from the beginning of the SMB2 header.

- An SMB2_REQ_GET_DFS_REFERRAL structure, as specified in [MS-DFSC] section 2.2.2, is constructed with the following parameters, and copied into the Buffer[]:
  - The MaxReferralLevel is set to the maximum referral level that the calling application will accept.
  - The path for which the referral information is being queried is copied into PathName[].
- The **InputCount** field is set to the size of the constructed SMB2_REQ_GET_DFS_REFERRAL structure.
- The **OutputOffset** field SHOULD be set to zero. <75>
- The **OutputCount** field is set to 0.
- The **MaxInputResponse** field is set to 0.
- The **MaxOutputResponse** field is set to the maximum response buffer size that the calling application will accept.
- SMB2_0_IOCTL_IS_FSCTL is set to TRUE in the Flags field.

The request MUST be sent to the server.

3.2.4.19.4 Application Requests a Pipe Transaction

The application MUST provide:

- The local identifier of the named pipe on which to issue the operation.
- An input buffer.
- A maximum output buffer response size, in bytes.

The client MUST locate the open in the GlobalOpenTable. If no open is found, the client MUST fail the application request.

If the open is found, the Open.Connection is NULL, and the Open.Durable is TRUE, the client MAY attempt to reconnect to this open, as specified in section 3.2.4.3.8. If the reconnect succeeds, this FSCTL MUST be retried. If it fails, the error code MUST be returned to the application.<76>

If the open is found, the Open.Connection is NULL, and the Open.Durable is FALSE, the client MUST fail this FSCTL operation.

If the open is found and the Open.Connection is not NULL, the client initializes an SMB2 IOCTL request following the syntax specified in section 2.2.31. The SMB2 header MUST be initialized as follows:

- The **Command** field is set to SMB2 IOCTL.
- A sequence number is allocated from Connection.SequenceWindow. If no sequence number is available, it MUST wait for one to become available. The client sets the MessageId to the sequence number received from Connection.SequenceWindow.
- The **SessionId** field is set to Open.TreeConnect.Session.SessionId.
- The **TreeId** field is set to Open.TreeConnect.TreeConnectId.

- The **ProcessId** field is set to a 4-byte identifier for the process that issued the IOCTL request.

The SMB2 IOCTL request MUST be initialized as follows:

- The **CtlCode** field is set to FSCTL_PIPE_TRANSCEIVE.
- The **FileId** field is set to Open.FileId.
- The **InputOffset** field is set to the offset to the Buffer[], in bytes, from the beginning of the SMB2 header.
- The **InputCount** field is set to the size, in bytes, of the application-provided input buffer.
- The **OutputOffset** field SHOULD be set to zero. <77>
- The **OutputCount** field is set to 0.
- The input buffer that is received from the application is copied into the request at InputOffset bytes from the beginning of the SMB2 header.
- The **MaxInputResponse** field is set to 0.
- The **MaxOutputResponse** field is set to the maximum output buffer size that the application will accept.
- SMB2_0_IOCTL_IS_FSCTL is set to TRUE in the Flags field.

The request MUST be sent to the server.

3.2.4.19.5 Application Requests a Peek at Pipe Data

The application MUST provide:

- The local identifier of the named pipe on which to issue the operation.
- The number of bytes to peek at in the pipe buffer.

The client MUST locate the open in the GlobalOpenTable. If no open is found, the client MUST fail the application request.

If the open is found, the Open.Connection is NULL, and the Open.Durable is TRUE, the client MAY attempt to reconnect to this open, as specified in section 3.2.4.3.8. If the reconnect succeeds, this FSCTL MUST be retried. If it fails, the error code MUST be returned to the application.<78>

If the open is found, the Open.Connection is NULL, and the Open.Durable is FALSE, the client MUST fail this FSCTL operation.

If the open is found and the Open.Connection is not NULL, the client initializes an SMB2 IOCTL request following the syntax specified in section 2.2.31. The SMB2 header MUST be initialized as follows:

- The Command field is set to SMB2 IOCTL.
- A sequence number is allocated from the Connection.SequenceWindow. If no sequence number is available, it MUST wait for one to become available. The client sets the MessageId to the sequence number that is received from the Connection.SequenceWindow.
- The SessionId field is set to Open.TreeConnect.Session.SessionId.

- The **TreeId** field is set to Open.TreeConnect.TreeConnectId.
- The ProcessId field is set to a 4-byte identifier for the process that issued the IOCTL request.

The SMB2 IOCTL request MUST be initialized as follows:

- The **CtlCode** field is set to FSCTL_PIPE_PEEK.
- The FileId field is set to Open.FileId.
- The InputOffset field is set to the offset to the Buffer[], in bytes, from the beginning of the SMB2 header.
- The InputCount field is set to 0.
- The OutputOffset field SHOULD be set to zero. <79>
- The OutputCount field is set to 0.
- The MaxInputResponse field is set to 0.
- The MaxOutputResponse field is set to the number of bytes that the client wants to peek at.
- SMB2_0_IOCTL_IS_FSCTL is set to TRUE in the Flags field.

The request MUST be sent to the server.

3.2.4.19.6 Application Requests a Pass-Through Operation

An SMB2 server MAY support pass-through operation requests. <80>

The application MUST provide:

- The local identifier of the open of a file or named pipe on which to issue the operation.
- An input buffer.
- An output buffer.
- A maximum input buffer response size, in bytes.
- A maximum output buffer response size, in bytes.
- An operation code.
- A Boolean indicating whether the operation is an FSCTL or an IOCTL.

The client MUST locate the open in the GlobalOpenTable. If no open is found, the client MUST fail the application request.

If the open is found, the Open.Connection is NULL, and the Open.Durable is TRUE, the client MAY attempt to reconnect to this open, as specified in section 3.2.4.3.8. If the reconnect succeeds, the FSCTL or IOCTL MUST be retried. If it fails, the error code MUST be returned to the application.<81>

If the open is found, the Open.Connection is NULL, and the Open.Durable is FALSE, the client MUST fail the FSCTL or IOCTL operation.

If the open is found and the Open.Connection is not NULL, the client initializes an SMB2 IOCTL request following the syntax specified in section 2.2.31. The SMB2 header MUST be initialized as follows:

- The Command field is set to SMB2 IOCTL.
- A sequence number is allocated from the Connection.SequenceWindow. If no sequence number is available, it MUST wait for one to become available. The client sets the MessageId to the sequence number that is received from the Connection.SequenceWindow.
- The SessionId field is set to Open.TreeConnect.Session.SessionId.
- The **TreeId** field is set to Open.TreeConnect.TreeConnectId.
- The ProcessId field is set to a 4-byte identifier for the process that issued the IOCTL request.

The SMB2 IOCTL request MUST be initialized as follows:

- The **CtlCode** field is set to the operation code that is received from the application.
- The **FileId** field is set to Open.FileId.
- The **InputOffset** field is set to the offset to the Buffer[], in bytes, from the beginning of the SMB2 header.
- The **InputCount** field is set to the size, in bytes, of the application-provided input buffer.
- The input buffer received from the application is copied into the request at InputOffset bytes from the beginning of the SMB2 header.
- The OutputOffset field SHOULD be set to zero. <82>
- The **OutputCount** field is set to the size, in bytes, of the application-provided output buffer.
- The **MaxInputResponse** field is set to the maximum output buffer size, in bytes, that the application will accept.
- The **MaxOutputResponse** field is set to the maximum output buffer size, in bytes, that the application will accept.
- If the operation is an FSCTL, SMB2_0_IOCTL_IS_FSCTL in the **Flags** field is set to TRUE. Otherwise, it is set to FALSE.

The request MUST be sent to the server.

3.2.4.20 Application Requests Unlocking of an Array of Byte Ranges

The application MUST provide:

- The local identifier of the open of a file or named pipe.
- An array of byte ranges to unlock. For each range, the application provides:
  - A starting offset.
  - A length, in bytes.

The client MUST locate this open in the GlobalOpenTable. If no open is found, the client MUST fail the application request.

If the open is found, the Open.Connection is NULL, and the Open.Durable is TRUE, the client MAY attempt to reconnect to this open as specified in section 3.2.4.3.8. If the reconnect succeeds, the unlock MUST be retried. If it fails, the error code MUST be returned to the application.<83>

If the open is found, the Open.Connection is NULL, and the Open.Durable is FALSE, the client MUST fail the unlock operation.

If the open is found and the Open.Connection is not NULL, the client initializes an SMB2 LOCK request following the syntax specified in section 2.2.26. The SMB2 header MUST be initialized as follows:

- The Command field is set to SMB2 LOCK.
- A sequence number is allocated from the Connection.SequenceWindow. If no sequence number is available, it MUST wait for one to become available. The client sets MessageId to the sequence number that is received from the Connection.SequenceWindow.
- The SessionId field is set to Open.TreeConnect.Session.SessionId.
- The **TreeId** field is set to Open.TreeConnect.TreeConnectId.
- The ProcessId field is set to a 4-byte identifier for the process that issued the IOCTL request.

The SMB2 LOCK request MUST be initialized as follows:

- The FileId field is set to Open.FileId.
- The LockCount field is set to the number of byte ranges being unlocked.
- For each range being unlocked, the client creates an SMB2_LOCK structure and places it in the Locks[] array of the request, setting the following values:
  - The offset is set to the offset of the range being unlocked.
  - The length is set to the length of the range to be unlocked.
  - The client sets SMB2_LOCKFLAG_UNLOCK to TRUE in the Flags field.

The request MUST be sent to the server.

3.2.4.21 Application Requests Closing a Share Connection

The application MUST provide:

- The server name.
- The share name.
- The security principal requesting the share be closed.

This operation is most often issued by the client when it is timing out idle connections; however, it is possible that a higher-level application may call this operation directly.

The client MUST locate the connection for this server name in the ConnectionTable. If the client supports multiple connections to the same server, it MUST repeat the remaining steps for each

connection to this server until it finds the session. When the tree connect is found, or if no tree connect is found, the operation MUST be completed successfully.

The client MUST attempt to locate a session for this security principal on this connection by walking the sessions in the Connection.SessionTable. If no session is found for this security principal, the client MUST continue on to the next connection.

The client MUST attempt to locate a tree connect for this share name in the Session.TreeConnectTable. If no tree connect is found, the client MUST continue on to the next connection. If a tree connect is found, the client MUST close all open files on this tree connect by walking the Connection.OpenTable, and issuing a close, as specified in section 3.2.4.4 for each open, where Open.TreeConnect is equal to the tree connect that is identified.

The client initializes an SMB2 TREE_DISCONNECT request following the syntax specified in section 2.2.11. The SMB2 header MUST be initialized as follows:

- The Command field is set to SMB2 TREE_DISCONNECT.
- A sequence number is allocated from the Connection.SequenceWindow. If no sequence number is available, it MUST wait for one to become available. The client sets MessageId to the sequence number that is received from the Connection.SequenceWindow.
- The SessionId field is set to TreeConnect.Session.SessionId.
- The **TreeId** field is set to TreeConnect.TreeConnectId.
- The ProcessId field is set to 0xFEFF.

The SMB2 TREE_DISCONNECT request MUST be initialized to the default values, as specified in 2.2.11.

The request MUST be sent to the server.

3.2.4.22 Application Requests Terminating an Authenticated Context

The application MUST provide:

- The server name.
- The security principal for the session that is being closed.

This operation is most often issued by the client when it is timing out idle sessions; however, it is possible that a higher-level application may call this operation directly.

The client MUST locate the connection for this server name in the ConnectionTable. If the client supports multiple connections to the same server, it MUST repeat the remaining steps for each connection to this server until the session is found. When the session is found, or if no session is found, the operation MUST be completed successfully.

The client MUST attempt to locate a session for this security principal on this connection by walking the sessions in the Connection.SessionTable. If no session is found for this security principal, the client MUST continue on to the next connection.

For the session found, the client MUST close all tree connects in the Session.TreeConnectTable, as specified in section 3.2.4.21.

The client initializes an SMB2 LOGOFF request following the syntax specified in section 2.2.7. The SMB2 header MUST be initialized as follows:

- The Command field is set to SMB2 LOGOFF.
- A sequence number is allocated from the Connection.SequenceWindow. If no sequence number is available, it MUST wait for one to become available. The client sets the MessageId to the sequence number that is received from the Connection.SequenceWindow.
- The SessionId field is set to Session.SessionId.
- The ProcessId field is set to 0xFEFF.

The SMB2 LOGOFF request MUST be initialized to the default values, as specified in 2.2.7.

The request MUST be sent to the server.

3.2.4.23 Application Requests Canceling an Operation

The application indicates the operation that is to be canceled.

The client MUST locate the connection on which the request was sent and find the MessageId for that request by locating it in Connection.OutstandingRequests.

The client initializes an SMB2 CANCEL request following the syntax specified in section 2.2.30. The SMB2 header MUST be initialized as follows:

- The Command field MUST be set to SMB2 CANCEL.
- The MessageId field is set to the identifier that is previously used for the request being canceled. Because the same MessageId is reused, cancel requests MUST NOT consume a sequence number.
- If the command has returned an indication of an asynchronous response, the client sets SMB2_FLAGS_ASYNC_COMMAND to TRUE in the Flags field and sets AsyncId to the asynchronous identifier for the request.

The SMB2 CANCEL request MUST be initialized to the default values, as specified in 2.2.30.

The request MUST be sent to the server.

There is no response to a cancel request. If the server has not processed the original request, it MUST cancel that pending request.

3.2.4.24 Application Requests the Session Key for an Authenticated Context

The application MUST provide:

- The server name.
- The security principal for the session whose key is being queried.

The client MUST locate the connection for this server name in the ConnectionTable. If the client supports multiple connections to the same server, it MUST repeat the remaining steps for each connection to this server until the session is found. If no session is found, the operation MUST be failed.

The client MUST attempt to locate a session for this security principal on this connection by walking the sessions in the Connection.SessionTable. If no session is found for this security principal, the client MUST continue on to the next connection.

If a session is found, the client MUST return the Session.SessionKey to the calling application.

3.2.5 Message Processing Events and Sequencing Rules

The SMB 2.0 Protocol client is driven by a series of response messages that are sent by the server. Processing for these messages is determined by the command in the SMB2 header of the response and is detailed for each of the SMB2 response messages in the sections that follow.

3.2.5.1 Receiving Any Message

Unless specifically noted in a subsequent section, the following logic MUST be applied to any response message that is received from the server by the client.

3.2.5.1.1 Finding the Application Request for This Response

The client MUST locate the request for which this response was sent in reply by locating the request in Connection.OutstandingRequests using the **MessageId** field of the SMB2 header. If the request is not found, the response MUST be discarded as invalid.

If the MessageId is 0xFFFFFFFFFFFFFFFF, this is not a reply to a previous request, and the client MUST NOT attempt to locate the request, but instead process it as follows:

If the command field in the SMB2 header is SMB2 OPLOCK_BREAK, it MUST be processed as specified in 3.2.5.19. Otherwise, the response MUST be discarded as invalid.

3.2.5.1.2 Verifying the Signature

If **Session.ShouldSign** is FALSE, or if MessageId is 0xFFFFFFFFFFFFFFFF, no verification is necessary.

If the SMB2 header of the response has SMB2_FLAGS_SIGNED set in the Flags field, the server MUST verify the signature as follows:

The server MUST look up the session in the Connection.SessionTable using the SessionId in the SMB2 header of the response. If the session is not found, the response MUST be discarded as invalid.

If the session is found, the server MUST verify the signature of the message as specified in section 3.1.5.1, using Session.SessionKey as the session key, and passing the response message.

If signature verification fails, the client MUST discard the received message and do no further processing for it. The client MAY also choose to disconnect the connection. If signature verification succeeds, the client MUST continue processing the packet, as specified in subsequent sections.

If the SMB2 header of the request does not have SMB2_FLAGS_SIGNED set in the Flags field, the client MUST determine if the server failed to sign a packet that required signing. The server MUST look up the session in the Connection.SessionTable using the SessionId in the SMB2 header of the request. If the session is found, and Session.ShouldSign is equal to TRUE, the client MUST discard the received message and do no further processing for it. The client MAY also choose to disconnect the connection. If there is no SessionId, if the session is not found, or if Session.ShouldSign is FALSE, the client continues processing on the packet, as specified in subsequent sections.<84>

3.2.5.1.3 Granting Message Credits

If CreditResponse is greater than 0, the client MUST insert the newly granted credits into the Connection.SequenceWindow. For each credit that is granted, the client MUST insert the next highest value into the sequence window, as specified in section 3.2.1.1. The client MUST then signal any requests that were waiting for available message identifiers to continue processing.

3.2.5.1.4 Handling Asynchronous Responses

If SMB2_FLAGS_ASYNC_COMMAND is set in the Flags field of the SMB2 header of the response and the Status field in the SMB2 header is STATUS_PENDING, the client MUST mark the request in Connection.OutstandingRequests as being handled asynchronously and store the new AsyncId of the request. Processing of this response is now complete.

If SMB2_FLAGS_ASYNC_COMMAND is set in the Flags field of the SMB2 header and Status is not STATUS_PENDING, this is a response to an asynchronous request and processing MUST continue as specified below.

3.2.5.1.5 Handling Session Expiration

If the Status field in the SMB2 header is STATUS_NETWORK_SESSION_EXPIRED, the client MUST attempt to reauthenticate the session that is identified by the SessionId in the SMB2 header, as specified in section 3.2.4.2.3. If the reauthentication attempt succeeds, the client MUST retry the request that failed with STATUS_NETWORK_SESSION_EXPIRED. If the reauthentication attempt fails, the client MUST fail the operation and terminate the session, as specified in section 3.2.4.22.

3.2.5.1.6 Handling Incorrectly Formatted Responses

If the client receives a response that does not conform to the structures specified in 2, the client MUST discard the response and fail the corresponding application request with an error indicating that an invalid network response was received. The client MAY also disconnect the connection.<85>

3.2.5.1.7 Processing the Response

The client MUST process the response based on the Command field of the SMB2 header of the response. When the processing is completed, the corresponding request MUST be removed from Connection.OutstandingRequests.

If the command that is received is not a valid command, or if the server returned a command that did not match the command of the request, the client SHOULD fail the application request with an implementation-specific error that indicates an invalid network response was received.<86>

3.2.5.1.8 Handling Compounded Responses

A client detects that a server sent a compounded response (multiple responses chained together into a single network send) by checking if the NextCommand in the SMB2 header of the response is not equal to 0. The client MUST handle compounded responses by separating them into individual responses.

For a series of responses compounded together, each response MUST be processed in order as an individual message with a size, in bytes, as determined by the NextCommand field in the SMB2 header. The final response in the compounded response chain will have NextCommand equal to 0, and it MUST be processed as an individual message of a size equal to the number of bytes remaining in this receive.

3.2.5.2 Receiving an SMB2 NEGOTIATE Response

If the Status field in the SMB2 header of the response is not STATUS_SUCCESS, the client MUST return the error code to the calling application.

Otherwise, the client MUST store the received MaxTransactSize in Connection.MaxTransactSize, the received MaxReadSize in Connection.MaxReadSize, the received MaxWriteSize in

Connection.MaxWriteSize, and the received ServerGuid in Connection.ServerGuid.<87> The client MUST store the received security buffer described by SecurityBufferOffset and SecurityBufferLength into Connection.GSSNegotiateToken.

If the SMB2_NEGOTIATE_SIGNING_REQUIRED bit in the SecurityMode field of the SMB2 NEGOTIATE response is set, the client MUST set Connection.RequireSigning to TRUE.

The client MUST continue processing, as specified in section 3.2.4.2.3.

3.2.5.3 Receiving an SMB2 SESSION_SETUP Response

The client MUST attempt to locate a session in Connection.SessionTable using the SessionId in the SMB2 header of the SMB2 SESSION_SETUP response. If a session is not located, this response MUST be handled as a new authentication attempt. If a session is located, this response MUST be handled as a reauthentication attempt.

3.2.5.3.1 Handling a New Authentication

If the **Status** field in the SMB2 header of the response is not STATUS_SUCCESS and is not STATUS_MORE_PROCESSING_REQUIRED, the client MUST return the error code to the calling application that initiated the authentication request and processing is complete.

Otherwise, the client MUST process the GSS token received in the SMB2 SESSION_SETUP response following the SMB2 header, described by SecurityBufferOffset and SecurityBufferLength. The client MUST use the configured GSS authentication protocol as specified in [MS-SPNG] section 3.3.5 and [RFC4178] section 3.2 to obtain the next GSS output token for the authentication exchange. Based on the result from the GSS authentication protocol, one of the following actions will be taken:

If the GSS protocol indicates an error, the error MUST be returned to the calling application that initiated the authentication request and processing is complete.

If the GSS protocol returns success, and the Status code of the SMB2 header of the response was STATUS_SUCCESS, authentication is complete. The client MUST allocate a session object and place it in the Connection.SessionTable. The session MUST be initialized as follows:

- Session.SessionId MUST be set to the SessionId in the SMB2 header of the response.
- Session.TreeConnectTable MUST be set to an empty table.
- Session.SessionKey MUST be set to the value queried from the GSS protocol, which queries it from the underlying authentication protocol. For information about how this is calculated for Kerberos authentication via Generic Security Service Application Programming Interface (GSS-API), see [MS-KILE] section 3.1.1.2. For information about how this is calculated for NTLM authentication via GSS-API, see [MS-NLMP] section 3.1.5.1.
- Session.Connection MUST be set to the connection on which this authentication attempt was issued.
- If the global setting RequireMessageSigning is set to TRUE or Connection.RequireSigning is set to TRUE then Session.ShouldSign MUST be set to TRUE, otherwise Session.ShouldSign MUST be set to FALSE.
- If the SMB2_SESSION_FLAG_IS_NULL bit is set in the **SessionFlags** field of the SMB2 SESSION_SETUP response, Session.ShouldSign MUST be set to FALSE.

- If the SMB2_SESSION_FLAG_IS_GUEST bit is set in the **SessionFlags** field of the SMB2 SESSION_SETUP response AND if Session.ShouldSign is TRUE, this indicates a SESSION_SETUP failure and the connection MUST be terminated.

The client MUST return success to the calling application that initiated the authentication request, and processing is complete.

If the GSS protocol returns success and the Status code of the SMB2 header of the response was STATUS_MORE_PROCESSING_REQUIRED, the client MUST send a subsequent session setup request to continue the authentication attempt. The client MUST construct an SMB2 SESSION_SETUP request by following the syntax specified in section 2.2.5. The SMB2 header MUST be initialized as follows:

- The **Command** field MUST be set to SMB2 SESSION_SETUP.
- The client MUST remove a sequence number from the Connection.SequenceWindow. If no sequence number is available, it MUST wait for one to become available. The client MUST set the MessageId field to the sequence number received from Connection.SequenceWindow.
- The client MUST set the **ProcessId** field to 0xFEFF.
- The client MUST set the **SessionId** field in the SMB2 header of the new request to the SessionId received in the SMB2 header of the response.

The SMB2 SESSION_SETUP request MUST be initialized as follows:

- The client MUST set the SMB2_NEGOTIATE_SIGNING_ENABLED bit in the **SecurityMode** field.
- If RequireMessageSigning is TRUE, the client MUST set the SMB2_NEGOTIATE_SIGNING_REQUIRED bit in the **SecurityMode** field.
- The client MUST set the **VcNumber** field to 0.
- If the client supports the Distributed File System (DFS), the client MUST set the SMB2_GLOBAL_CAP_DFS bit in the **Capabilities** field. For more information about DFS, see [MSDFS].
- The client MUST copy the GSS output token into the response. The client MUST set SecurityBufferOffset and SecurityBufferLength to describe the GSS output token.

This request MUST be sent to the server.

3.2.5.3.2 Handling a Reauthentication

If the **Status** field in the SMB2 header of the response is not STATUS_SUCCESS and is not STATUS_MORE_PROCESSING_REQUIRED, the client MUST return the error code to the calling application that initiated the reauthentication request and processing is complete.

Otherwise, the client MUST process the Generic Security Service (GSS) token that is received in the SMB2 SESSION_SETUP response following the SMB2 header, described by SecurityBufferOffset and SecurityBufferLength. The client MUST use the configured GSS authentication protocol, as specified in [MS-SPNG] section 3.3.5 and [RFC4178] section 3.2, to obtain the next GSS output token for the authentication exchange. Based on the result from the GSS authentication protocol, one of the following actions will be taken:

If the GSS protocol indicates an error, the error MUST be returned to the calling application that initiated the reauthentication request and processing is complete.

If the GSS protocol returns success and the Status code of the SMB2 header of the response was STATUS_SUCCESS, reauthentication is complete. The client MUST return success to the calling application that initiated the reauthentication request.

If the GSS protocol returns success and the Status code of the SMB2 header of the response was STATUS_MORE_PROCESSING_REQUIRED, the client MUST send a subsequent session setup request to continue the reauthentication attempt. The client MUST construct an SMB2 SESSION_SETUP request following the syntax specified in section 2.2.5. The SMB2 header MUST be initialized as follows:

- The **Command** field MUST be set to SMB2 SESSION_SETUP.
- The client MUST remove a sequence number from Connection.SequenceWindow. If no sequence number is available, it MUST wait for one to become available. The client MUST set the **MessageId** field to the sequence number that is received from Connection.SequenceWindow.
- The client MUST set the **ProcessId** field to 0xFEFF.
- The client MUST set the **SessionId** field in the SMB2 header of the new request to the SessionId received in the SMB2 header of the response.
- The client MUST NOT regenerate Session.SessionKey.

The SMB2 SESSION_SETUP request MUST be initialized as follows:

- The client MUST set the SMB2_NEGOTIATE_SIGNING_ENABLED bit in the **SecurityMode** field.
- If RequireMessageSigning is TRUE, the client MUST set the SMB2_NEGOTIATE_SIGNING_REQUIRED bit in the **SecurityMode** field.
- The client MUST set the **VcNumber** field to 0.
- If the client supports the Distributed File System (DFS), the client MUST set the SMB2_GLOBAL_CAP_DFS bit in the **Capabilities** field. For more information about DFS, see [MSDFS].
- The client MUST copy the GSS output token into the response. The client MUST set SecurityBufferOffset and SecurityBufferLength to describe the GSS output token.

This request MUST be sent to the server.

3.2.5.4 Receiving an SMB2 LOGOFF Response

The client MUST locate the session in Connection.SessionTable using the SessionId in the SMB2 header of the response. The associated session object MUST be removed from the Connection.SessionTable and freed. The client MUST return success to the application that requested the authenticated context termination.

3.2.5.5 Receiving an SMB2 TREE_CONNECT Response

If the Status field of the SMB2 header of the response indicates an error, the client MUST return the received status code to the calling application.

If the Status field of the SMB2 header of the response indicates success, the client MUST locate the session in the Connection.SessionTable using the SessionId in the SMB2 header of the response. The client MUST allocate a tree connect object and insert it into the Session.TreeConnectTable. The tree connect is initialized as follows:

- The TreeConnect.TreeConnectId MUST be set to the **TreeId** that is received in the SMB2 header of the response.
- The TreeConnect.MaximalAccess MUST be set to the MaximalAccess received in the SMB2 TREE_CONNECT response following the SMB2 header.
- The TreeConnect.Session MUST be set to the session that is looked up using SessionId from the SMB2 header of the response.

The client MUST return success to the application that initiated the connection to the share.

3.2.5.6 Receiving an SMB2 TREE_DISCONNECT Response

The client MUST locate the session in the Connection.SessionTable using the SessionId in the SMB2 header of the response. It MUST locate the tree connect in the Session.TreeConnectTable using the **TreeId** in the SMB2 header of the response. The associated tree connect object MUST be removed from the Session.TreeConnectTable and freed. The client MUST return success to the application that requested the share connection termination.

3.2.5.7 Receiving an SMB2 CREATE Response for a New Create Operation

If the Status field of the SMB2 header of the response indicates an error, the client MUST return the received status code to the calling application that initiated the open of a file or named pipe.

If the Status field of the SMB2 header of the response indicates success, the client MUST locate the session in the Connection.SessionTable using the SessionId in the SMB2 header of the response. The client MUST locate a tree connect in the Session.TreeConnectTable using the **TreeId** in the SMB2 header of the response. The client MUST allocate an open object and initialize it as follows:

- Open.ClientLocalId MUST be set to a locally generated identifier that the calling application will use to refer to this open.
- Open.FileId MUST be set to the FileId that is received in the SMB2 CREATE response following the SMB2 header.
- Open.TreeConnect MUST be set to the tree connect that was looked up in the Session.TreeConnectTable.
- Open.Connection MUST be set to the connection on which the response was received.
- Open.OplockLevel MUST be set to the OplockLevel in the SMB2 CREATE response.
- Open.Durable MUST be set to FALSE.
- Open.MaximalAccess MUST be set to 0.

If the response includes response create contexts following the syntax specified in section 2.2.14.2, the processing described in subsequent subsections MUST be handled if the specified create context is present in the response.

The client MUST insert the open into the Connection.OpenTable and GlobalOpenTable. The client MUST return success and Open.ClientLocalId to the application that initiated the create operation.

3.2.5.7.1 SMB2_CREATE_DURABLE_HANDLE_REQUEST Create Context

If the SMB2_CREATE_DURABLE_HANDLE_REQUEST context is present, the client MUST set Open.Durable to TRUE.

3.2.5.7.2 SMB2_CREATE_QUERY_MAXIMAL_ACCESS Create Context

If the SMB2_CREATE_QUERY_MAXIMAL_ACCESS context is present, and QueryStatus in the SMB2_CREATE_QUERY_MAXIMAL_ACCESS context is STATUS_SUCCESS, the client MUST set Open.MaximalAccess to the MaximalAccess received in the context.

3.2.5.8 Receiving an SMB2 CREATE Response for an Open Reestablishment

If the Status field of the SMB2 header of the response indicates an error, the client MUST return the received status code to the calling application that initiated the open of a file or named pipe.

If the Status field of the SMB2 header of the response indicates success, the client MUST locate the open in the GlobalOpenTable using the persistent portion of the FileId returned in the SMB2 CREATE response. The following fields MUST be reinitialized:

- Open.FileId MUST be set to the FileId received in the SMB2 CREATE response following the SMB2 header.
- Open.TreeConnect MUST be set to the tree connect that was looked up in the Session.TreeConnectTable.
- Open.Connection MUST be set to the connection on which the response was received.

The client MUST insert the open into the Connection.OpenTable. The client MUST return success to the application that initiated the open reestablishment operation.

3.2.5.9 Receiving an SMB2 CLOSE Response

The client MUST locate the open in the Connection.OpenTable using the FileId in the SMB2 header of the response. The open object MUST be removed from the Connection.OpenTable and the GlobalOpenTable and freed. The client MUST return success to the application that requested the file or named pipe be closed.

3.2.5.10 Receiving an SMB2 FLUSH Response

The client MUST return the received status code in the Status field of the SMB2 header of the response to the application that issued the request to flush data on the file or named pipe.

3.2.5.11 Receiving an SMB2 READ Response

If the Status field of the SMB2 header of the response indicates an error, the client MUST return the received status code to the calling application.

If the Status field of the SMB2 header of the response indicates success, the client MUST copy the received information in the SMB2 READ response following the SMB2 header described by DataOffset and DataLength into the buffer that is provided by the calling application. The client MUST return success and DataLength to the application.

3.2.5.12 Receiving an SMB2 WRITE Response

The client MUST return the received status code in the Status field of the SMB2 header of the response to the application that issued the request to write data to the file or named pipe. The client MUST also return the Count value from the SMB2 WRITE response following the SMB2 header, indicating how many bytes were written.

3.2.5.13 Receiving an SMB2 LOCK Response

The client MUST return the received status code in the Status field of the SMB2 header of the response to the application that issued the request to lock or unlock ranges on the file.

3.2.5.14 Receiving an SMB2 IOCTL Response

3.2.5.14.1 Handling an Enumeration of Previous Versions Response

If the Status field of the SMB2 header of the response indicates an error, the client MUST return the received status code to the calling application.

If the Status field of the SMB2 header of the response indicates success, the client MUST copy the received information in the SMB2 IOCTL response following the SMB2 header described by OutputOffset and OutputLength into the buffer that is provided by the calling application for receiving the response output buffer. The client MUST return success and OutputLength to the application.

3.2.5.14.2 Handling a Server-Side Data Copy Source File Key Response

If the Status field of the SMB2 header of the response indicates an error, the client MUST return the received status code to the calling application.

If the Status field of the SMB2 header of the response indicates success, the client MUST copy the received information in the SMB2 IOCTL response following the SMB2 header described by OutputOffset and OutputLength into the buffer that is provided by the calling application for receiving the response output buffer. The client MUST return success and OutputLength to the application.

3.2.5.14.3 Handling a Server-Side Data Copy Response

If the Status field of the SMB2 header of the response indicates an error, the client MUST return the received status code to the calling application. If the status code is STATUS_INVALID_PARAMETER and the StructureSize of the response indicates that the server has provided an SRV_COPYCHUNK_RESPONSE, the client SHOULD return the values in the SRV_COPYCHUNK_RESPONSE to the application indicating the maximum limits the server supports for server side copy operations.

If the Status field of the SMB2 header of the response indicates success, the client MUST copy the received information in the SMB2 IOCTL response following the SMB2 header that is described by OutputOffset and OutputLength into the buffer that is provided by the calling application for receiving the response output buffer. The client MUST return success and the OutputLength to the application.

3.2.5.14.4 Handling a DFS Referral Information Response

If the Status field of the SMB2 header of the response indicates an error, the client MUST return the received status code to the calling application.

If the Status field of the SMB2 header of the response indicates success, the client MUST copy the received information in the SMB2 IOCTL response following the SMB2 header that is described by OutputOffset and OutputLength into the buffer that is provided by the calling application for receiving the response output buffer. The client MUST return success and the OutputLength to the application.

3.2.5.14.5 Handling a Pipe Transaction Response

If the Status field of the SMB2 header of the response indicates an error, the client MUST return the received status code to the calling application.

If the Status field of the SMB2 header of the response indicates success, the client MUST copy the received information in the SMB2 IOCTL response following the SMB2 header that is described by the OutputOffset and OutputCount into the buffer that is provided by the calling application for receiving the response output buffer. The client MUST ignore the InputOffset and InputCount. The client MUST return success and the OutputLength to the application.

3.2.5.14.6 Handling a Peek at Pipe Data Response

If the Status field of the SMB2 header of the response indicates an error, the client MUST return the received status code to the calling application.

If the Status field of the SMB2 header of the response indicates success, the client MUST copy the received information in the SMB2 IOCTL response following the SMB2 header that is described by the OutputOffset and OutputLength into the buffer that is provided by the calling application for receiving the response output buffer. The client MUST return success and the OutputLength to the application.

3.2.5.14.7 Handling a Pass-Through Operation Response

If the Status field of the SMB2 header of the response indicates an error, the client MUST return the received status code to the calling application.

If the Status field of the SMB2 header of the response indicates success, the client MUST copy the received information in the SMB2 IOCTL response following the SMB2 header that is described by the InputOffset and InputLength into the buffer that is provided by the calling application for receiving the response input buffer. The client MUST copy the received information in the SMB2 IOCTL response following the SMB2 header that is described by the OutputOffset and OutputLength into the buffer that is provided by the calling application for receiving the response output buffer. The client MUST return success, the InputLength, and the OutputLength to the application.

3.2.5.15 Receiving an SMB2 QUERY_DIRECTORY Response

If the Status field of the SMB2 header of the response indicates an error, the client MUST return the received status code to the calling application.

If the Status field of the SMB2 header of the response indicates success, the client MUST copy the received information in the SMB2 QUERY_DIRECTORY response following the SMB2 header that is described by the OutputBufferOffset and OutputBufferLength into the buffer that is provided by the calling application. The client MUST return success and the OutputBufferLength to the application.

3.2.5.16 Receiving an SMB2 CHANGE_NOTIFY Response

If the Status field of the SMB2 header of the response indicates an error, the client MUST return the received status code to the calling application.

If the Status field of the SMB2 header of the response indicates success, the client MUST copy the received information in the SMB2 CHANGE_NOTIFY response following the SMB2 header that is described by the OutputBufferOffset and OutputBufferLength into the buffer that is provided by the calling application. The client MUST return success and the OutputBufferLength to the application.

3.2.5.17 Receiving an SMB2 QUERY_INFO Response

If the Status field of the SMB2 header of the response indicates an error, the client MUST return the received status code to the calling application. If the error code is STATUS_BUFFER_TOO_SMALL and the SMB2 ERROR response following the SMB2 header has a ByteCount of 4, the client MUST also return the 4-byte error data to the calling application. This error data MUST indicate the size, in bytes, that is required to successfully query the information.

If the Status field of the SMB2 header of the response indicates success, the client MUST copy the received information in the SMB2 QUERY_INFO response following the SMB2 header that is described by the OutputBufferOffset and OutputBufferLength into the buffer that is provided by the calling application. The client MUST return success and the OutputBufferLength to the application.

3.2.5.18 Receiving an SMB2 SET_INFO Response

The client MUST return the received status code in the Status field of the SMB2 header of the response to the application that issued the request to set information on the file, file system, or named pipe. This applies for requests to set file information, file system information, quota information, and security information.

3.2.5.19 Receiving an SMB2 OPLOCK_BREAK Notification

If the MessageId field of the SMB2 header of the response is 0xFFFFFFFFFFFFFFFF, this MUST be processed as an oplock break indication. Otherwise, the client MUST process it as a response to an oplock break acknowledgment.

3.2.5.19.1 Receiving an Oplock Break Indication

The client MUST locate the open in the Connection.OpenTable using the FileId in the SMB2 OPLOCK BREAK Notification following the SMB2 header. If the open is not found, the oplock break indication MUST be discarded, and no further processing is required.

If the open is found, the client MUST take action based on the Open.OplockLevel and the new OplockLevel that is received in the SMB2 OPLOCK_BREAK response:

If the Open.OplockLevel is SMB2_OPLOCK_LEVEL_NONE, no action is required, and no further processing is required.

If the Open.OplockLevel is SMB2_OPLOCK_LEVEL_II, and the OplockLevel is SMB2_OPLOCK_LEVEL_NONE, the client MUST set Open.OplockLevel to SMB2_OPLOCK_LEVEL_NONE.

If the Open.OplockLevel is SMB2_OPLOCK_LEVEL_BATCH, and the OplockLevel is SMB2_OPLOCK_LEVEL_NONE, the client MUST set Open.OplockLevel to SMB2_OPLOCK_LEVEL_NONE. The client MUST flush any writes or byte range locks that it has cached locally to the server. When that is complete, the client MUST send an oplock break acknowledgment, as specified in the following sections.

If the Open.OplockLevel is SMB2_OPLOCK_LEVEL_BATCH, and the OplockLevel is SMB2_OPLOCK_LEVEL_II, the client MUST set Open.OplockLevel to SMB2_OPLOCK_LEVEL_II. The client MUST flush any writes or byte range locks that it has cached locally to the server. When that is complete, the client MUST send an oplock break acknowledgment, as specified below.

The client MAY choose to request and support SMB2_OPLOCK_LEVEL_EXCLUSIVE. If it does, the break operation would match those specified above for SMB2_OPLOCK_LEVEL_BATCH. It MUST NOT break from batch to exclusive.<88>

If the client is required to send an oplock break acknowledgment, it MUST construct a request following the syntax that is specified in section 2.2.24. The SMB2 header is initialized as follows:

- Command MUST be set to SMB2 OPLOCK_BREAK.
- The client MUST remove a sequence number from Connection.SequenceWindow. If no sequence number is available, it MUST wait for one to become available. The client MUST set MessageId to the sequence number that is received from Connection.SequenceWindow.
- The client MUST set SessionId to Open.TreeConnect.Session.SessionId.
- The client MUST set **TreeId** to Open.TreeConnect.TreeConnectId.
- The client MUST set ProcessId to 0xFEFF.

The SMB2 OPLOCK_BREAK request is initialized as follows:

- The FileId MUST be set to Open.FileId.
- The OplockLevel MUST be set to Open.OplockLevel.

The request MUST be sent to the server.

3.2.5.19.2 Receiving an Oplock Break Acknowledgment Response

No processing is required for this response.

3.2.6 Timer Events

3.2.6.1 Request Expiration Timer Event

When the Request Expiration timer expires, the client MUST walk all connections in the ConnectionTable. For each connection, the client MUST walk the outstanding requests in Connection.OutstandingRequests. If any request has exceeded its time-out interval and the server has not returned STATUS_PENDING (indicating asynchronous and possibly lengthy processing time, as specified in section 3.2.5.1.4), the client MAY<89> process the request as if it received a failure response and return an error to the calling application. The client MAY<90> choose to disconnect the connection as well.

3.2.6.2 Idle Connection Timer Event

When the Idle Connection timer expires, the client MUST walk all connections in the ConnectionTable. For any connection in which the Connection.OpenTable is empty and no operations have been performed within the default time-out, the client MAY choose to disconnect the connection.<91>

3.2.7 Other Local Events

3.2.7.1 Handling a Network Disconnect

When the underlying transport indicates a disconnect, the client MUST walk all opens in the Connection.OpenTable. For each open, if Open.Durable is not TRUE, the open MUST be removed from the Connection.OpenTable and the GlobalOpenTable, and freed.

If Open.Durable is TRUE, the open MUST be removed from the Connection.OpenTable, the Open.Connection MUST be set to NULL, and the Open.TreeConnect MUST be set to NULL.

The client MUST then walk all the sessions in the Connection.SessionTable. For each session, the client MUST walk all tree connects in the Session.TreeConnectTable. Each tree connect in the table MUST be freed. Then the session MUST be freed.

Finally, the connection MUST be removed from the ConnectionTable and freed.

3.3 Server Details

3.3.1 Abstract Data Model

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

3.3.1.1 Algorithm for Generating Async Identifiers

The server MUST implement an algorithm for the generation of asynchronous request identifiers for an SMB2 request that is being handled asynchronously. The algorithm MUST meet the following conditions:

- The identifier MUST be an 8-byte value.
- The identifier MUST be unique for all outstanding asynchronous requests on a specified SMB2 transport connection.
- The identifier MUST remain valid until the final response for the request is sent.
- The identifier MUST NOT be reused until the final response is sent.

3.3.1.2 Algorithm for Handling Available Message Sequence Numbers by the Server

The server MUST implement an algorithm to manage message sequence numbers. Sequence numbers are used to associate requests with responses, and to determine what requests are allowed for processing. The algorithm MUST meet the following conditions:

- When an SMB2 transport connection is first established, the allowable sequence numbers that comprise the valid command window for received messages on that connection MUST be the set { 0 }.
- After a sequence number is received, its value MUST never be allowed to be received again. (After the sequence number 0 is received, no other request that uses the sequence number 0 shall be processed.) If the 64-bit sequence wraps, the connection MUST be terminated.
- As credits are granted as specified in section 3.3.1.3, the acceptable sequence numbers MUST progress in a monotonically increasing manner. For example, if the set consists of { 0 }, and 3 credits are granted, the valid command window set MUST grow to { 0, 1, 2, 3 }.
- The server MUST allow requests to be received out of sequence. For example, if the valid command window set is { 0, 1, 2, 3 }, it is valid to receive a request with sequence number 2 before receiving a request with sequence number 0.
- The server MAY limit the maximum range of the acceptable sequence numbers. For example, if the valid command window set is { 0, 1, 2, 3, 4, 5 }, and the server receives requests for 1, 2, 3,

4, and 5, it MAY choose to not grant more credits and keep the valid command window set at { 0 } until the sequence number 0 is received.<92>

For the client side of this algorithm, see section 3.2.1.1.

3.3.1.3 Algorithm for the Granting of Credits

The server MUST implement an algorithm for granting credits to the client. Each credit provides the client the capability to send a request to the server. Multiple credits allow for multiple simultaneous requests. The algorithm MUST meet the following conditions:

- The number of credits granted to the client MUST be considered as 1 when the connection is established.
- The server MUST ensure that the number of credits granted to the client is never reduced to zero. If the condition occurs, there is no way for the client to send subsequent requests for more credits.
- The server MAY grant any number of credits up to that which the client requests.<93>
- The server MAY vary the number of credits granted to different clients based on quality of service features, such as identity, behavior, or administrator configuration.<94>

3.3.1.4 Algorithm for Change Notifications in an Object Store

The server MUST implement an algorithm that monitors for changes on an object store. The effect of this algorithm MUST be identical to that used to offer the behavior specified in [CIFS] section 4.3.7. The algorithm MUST meet the following conditions:

- If a change notification request is pending on a directory AND a change occurs to the directory contents matching the events to be monitored as specified in **CompletionFilter**, the server MUST copy the results into the buffer of the Change Notification response. The server MAY choose to aggregate one or more changes indicated by the underlying object store into a single response. The server MUST construct a CHANGE_NOTIFY response as specified in section 2.2.36. The server MUST then return the results to the client.
- The server SHOULD try to fit in as many events that match the **CompletionFilter** of the request as possible before completing the request.
- If a client issues multiple change notification requests on the same open to a directory, the server MUST queue the requests and complete them on a First In, First Out (FIFO) basis when changes are indicated by the underlying object store.
- If the client requested that an entire tree be watched, the server MUST monitor all objects beneath the directory on which the operation was issued, instead of simply the immediate children of that directory.
- Change notification information that is returned to the user MUST conform to the syntax specified in section 2.2.36.1.

3.3.1.5 Global Structures

ShareList: A list of available shares for the system. The structure of a share is as specified in section 3.3.1.6. The list MUST be uniquely indexed by the share name.

GlobalOpenTable: A table containing all the files opened by remote clients on the server. The structure of an open is as specified in section 3.3.1.10. The table MUST be uniquely indexed by **Open.DurableFileId** and MUST support enumeration of all entries in the table.

GlobalSessionTable: A list of all the authenticated sessions established to this server, indexed by the *SessionId*. The server MUST also be able to search the list by security principal, and the list MUST allow for multiple sessions with the same security principal on different connections.

ConnectionList: A list of all open connections on the server, indexed by the connection endpoint addresses.

ServerGuid: A global identifier for this server.

ServerStartTime: The start time of the SMB2 server, in FILETIME format as specified in [MS-DTYP] section 2.3.1.

3.3.1.6 Per Share Structures

Share.Name: A unique name for the shared resource on this server.

Share.LocalPath: A path that describes the local resource that is being shared. This MUST be a store that either provides named pipe functionality, or that offers storage and/or retrieval of files. In the case of the latter, it MAY be a device that accepts a file and then processes it in some format, such as a printer.<95>

Share.ConnectSecurity: An authorization policy such as an access control list that describes which users are allowed to connect to this share.

Share.FileSecurity: An authorization policy such as an access control list that describes what actions users that connect to this share are allowed to perform on the shared resource.<96>

Share.CscFlags: The configured offline caching policy for this share. This value MUST be manual caching, automatic caching of files, automatic caching of files and programs, or no offline caching. For more information, see section 2.2.10. For more information about offline caching, see [OFFLINE].

Share.IsDfs: A Boolean that, if set, indicates that this share is configured for DFS. For more information, see [MSDFS].

Share.DoAccessBasedDirectoryEnumeration: A Boolean that, if set, indicates that the results of directory enumerations on this share must be trimmed to include only the files and directories that the calling user has access to.

Share.AllowNamespaceCaching: A Boolean that, if set, indicates that clients are allowed to cache directory enumeration results for better performance.

Share.ForceSharedDelete: A Boolean that, if set, indicates that all opens on this share MUST include FILE_SHARE_DELETE in the sharing access.

Share.RestrictExclusiveOpens: A Boolean that, if set, indicates that users who request read-only access to a file are not allowed to deny other readers.

3.3.1.7 Per Transport Connection Structures

Connection.SessionTable: A table of authenticated sessions that have been established on this transport connection. The table MUST be uniquely indexed by **Session.SessionId**, and MUST support enumeration of every entry in the table.

Connection.CommandSequenceWindow: A list of the sequence numbers that are valid to receive from the client at this time. For more information, see section 3.3.1.2.

Connection.RequestList: A list of all client requests being processed. Each request MUST include a Boolean value that indicates whether it is being handled asynchronously.

Connection.ClientCapabilities: The capabilities of the client of this connection in a form that MUST follow the syntax as specified in section 2.2.5.

Connection.NegotiateReceived: A Boolean indicating whether a negotiate request has been received on this transport connection.

Connection.AsyncCommandList: A list of client requests being handled asynchronously. Each request MUST have been assigned an **AsyncId**.

3.3.1.8 Per Session Structures

Session.SessionId: A numeric value that uniquely identifies the session within the scope of the transport connection over which the session was established. This value, transformed into a 64-bit number, is typically sent to clients as the SessionId in the SMB2 header.

Session.Connection: The connection on which this session was established (see also sections 3.3.5.5.1 and 3.3.4.4).

Session.State: The current activity state of this session. This value MUST be either InProgress, Valid, or Expired.

Session.SecurityContext: The security context of the user that authenticated this session. This value MUST be in a form that allows for evaluating security descriptors within the server, as well as being passed to the underlying object store to handle security evaluation that may happen there.

Session.SessionKey: The 16-byte cryptographic key for this authenticated context.

Session.ShouldSign: A Boolean that, if set, indicates that this session MUST sign communication if signing is enabled on this connection.

Session.OpenTable: A table of opens of files or named pipes, as specified in section 3.3.1.10, that have been opened by this authenticated session. This table MUST be uniquely indexed by **Open.FileId** and MUST support enumeration of all entries in the table.

Session.TreeConnectTable: A table of tree connects that have been established by this authenticated session to shares on this server. This table MUST be uniquely indexed by **TreeConnect.TreeId** and MUST allow enumeration of all entries in the table.

Session.ExpirationTime: A value that specifies the time after which the client must reauthenticate with the server.

3.3.1.9 Per Tree Connect Structures

TreeConnect.TreeId: A numeric value that uniquely identifies a tree connect within the scope of the session over which it was established. This value is typically represented as a 32-bit **TreeId** in the SMB2 header.

TreeConnect.Session: A pointer to the authenticated session that established this tree connect.

TreeConnect.Share: A pointer to the share that this tree connect was established for.

3.3.1.10 Per Open Structures

Open.FileId: A numeric value that uniquely identifies the open handle to a file or a pipe within the scope of a session over which the handle was opened. This value is the volatile portion of the identifier. A 64-bit representation of this value, combined with **Open.DurableFileId** as described below, combine to form the **SMB2_FILEID** described in section 2.2.14.1.

Open.DurableFileId: A numeric value that uniquely identifies the open handle to a file or a pipe within the scope of all opens granted by the server, as described by the **GlobalOpenTable**. A 64-bit representation of this value combined with **Open.FileId**, as described above, form the **SMB2_FILEID** described in section 2.2.14.1. This value is the persistent portion of the identifier.

Open.Session: A reference to the authenticated session, as specified in section 3.3.1.8, over which this open was performed. If the file is not attached to a session at this time, its value MUST be 0.

Open.Connection: A reference to the connection, as specified in section 3.3.1.7, that created this open. If the file is not attached to a connection at this time, this value MUST be 0.

Open.LocalOpen: An open of a file or named pipe in the underlying local resource that is used to perform the local operations, such as reading or writing, to the underlying object.

Open.GrantedAccess: The access granted on this open, as defined in section 2.2.13.1.

Open.OplockLevel: The current oplock level for this open. This value MUST be either None, Level2, Exclusive, Batch, or Directory.

Open.OplockState: The current oplock state of the file. This value MUST be Held, Breaking, or None.

Open.OplockTimeout: The time-out value that indicates when an oplock that is breaking and has not received an acknowledgment from the client will be acknowledged by the server.

Open.IsDurable: A Boolean that indicates whether this open has requested durable operation.

Open.DurableOpenTimeout: A time-out value that indicates when a handle that has been preserved for durability will be closed by the system if a client has not reclaimed it.

Open.DurableOwner: A security descriptor that holds the original opener of the file. This allows the server to determine if a caller that is trying to reestablish a durable open is allowed to do so.

Open.EnumerationLocation: For directories, this value indicates the current location in a directory enumeration and allows for the continuing of an enumeration across multiple requests. For files, this value is unused.

Open.EnumerationSearchPattern: For directories, this value holds the search pattern that is used in directory enumeration and allows for the continuing of an enumeration across multiple requests. For files, this value is unused.

3.3.2 Timers

3.3.2.1 Oplock Break Acknowledgment Timer

This timer controls the amount of time the server waits for an oplock break acknowledgment from the client (as specified in section 2.2.24) after sending an oplock break notification (as specified in section 2.2.23) to the client. The server MUST wait for an interval of time greater than or equal to the oplock break acknowledgment timer. This timer MUST be smaller than the client Request Expiration time, as specified in section 3.2.6.1.<97>

3.3.2.2 Durable Open Scavenger Timer

This timer controls the amount of time the server keeps a durable handle active after the underlying transport connection to the client is lost.<98> The server MUST keep the durable handle active for at least this amount of time, except in the case where the underlying object store indicates an oplock break for this file. In this case the server MAY tear down the durable open. The server MAY also tear down a durable handle based on administrative action or resource constraints before the timer expires.

3.3.2.3 Session Expiration Timer

This timer controls the periodic scheduling of searching for sessions that have passed their expiration time. The server SHOULD schedule this timer such that sessions are expired in a timely manner.

3.3.3 Initialization

The server MUST establish a listening TCP endpoint on port number 445 as specified in [MS-SMB] section 2.1.1. The server MAY also listen on the NetBIOS transport as specified in [CIFS] section 2.6.2.<99>

GlobalOpenTable MUST be set to an empty table.

GlobalSessionTable MUST be set to an empty table.

ServerGuid MUST be set to an appropriately generated unique value. <100>

ConnectionList MUST be set to an empty list.

ServerStartTime MUST be set to the time at which the SMB2 server was started.

The ShareList MUST be populated based on server configuration. The values are retrieved from a persistent configuration store, and for each entry, a share MUST be created and inserted into the list. The information retrieved from the configuration MUST include:

- **Share.Name** is set to the Unicode name. The share name MUST be no longer than 80 characters (not including the NULL terminator) and MUST not contain any invalid characters, as specified in [CIFS] section 3.2.
- **Share.LocalPath** MUST correctly specify a local resource on the server computer, such as a local file system volume or a subdirectory on that volume.
- **Share.ConnectSecurity** MUST be set to the security descriptor provided in the configuration store. If one is not provided, it MUST be set to a descriptor that allows all users.
- **Share.FileSecurity** MUST be set to the security descriptor provided in the configuration store. If one is not provided, it MUST be set to a descriptor that allows all users.
- **Share.CscFlags** MUST be set to the value provided in the configuration store. If one is not provided, it MUST be set to manual caching.
- **Share.IsDfs** MUST be set to the value provided in the configuration store. If one is not provided, it MUST be set to FALSE.
- **Share.DoAccessBasedDirectoryEnumeration** MUST be set to the value provided in the configuration store. If one is not provided, it MUST be set to FALSE.

- **Share.AllowNamespaceCaching** MUST be set to the value provided in the configuration store. If one is not provided, it MUST be set to FALSE.
- **Share.ForceSharedDelete** MUST be set to the value provided in the configuration store. If one is not provided, it MUST be set to FALSE.
- **Share.RestrictExclusiveOpens** MUST be set to the value provided in the configuration store. If one is not provided, it MUST be set to FALSE.

3.3.4 Higher-Layer Triggered Events

The SMB 2.0 Protocol server is driven by a series of higher-layer triggered events in the following categories:

- Indications of buffering state changes on local opens (oplock breaks).
- Requests for the session key of authenticated sessions.
- Required actions for sending any outgoing message.

The following sections provide details on the above events.

3.3.4.1 Sending Any Outgoing Message

Unless specifically noted in a subsequent section, the following logic MUST be applied to any response message being sent from the server to the client.

3.3.4.1.1 Signing the Message

If the response message being sent contains a SessionId that is nonzero in the SMB2 header field, and the session in **Connection.SessionTable** identified by that identifier has **Session.ShouldSign** equal to TRUE, and the MessageId in the SMB2 header is not 0xFFFFFFFFFFFFFFFF, the response MUST be signed as specified below. Otherwise, the packet MUST NOT be signed.

The server MUST sign the response as specified in section 3.1.4.1. The server provides the **Session.SessionKey**, the length of the response, and the response itself, and calculates the signature as described.

3.3.4.1.2 Granting Credits to the Client

As described in section 3.3.1.2, the server maintains a list of message identifiers available for incoming requests. The total number of available message identifiers can change dynamically as the system runs, with the server granting credits based on some local policy.

Based on the CreditRequest specified in the SMB2 header of a client request, the server MUST determine how many credits it will grant the client on each request by using a vendor-specific algorithm as specified in section 3.3.1.3. The server MUST then place the number of credits granted in the CreditResponse field in the SMB2 header of the response.

To maintain the same number of credits already granted, the server returns a value of 1, which simply returns the credit consumed by this command. To reduce the number of credits granted, the server returns a value of 0 which implies that the credit consumed by this command is not returned to the client. To increase the number of credits granted, the server returns a value greater than 1.

Asynchronous responses, as specified in section 3.3.4.2, MAY grant credits.<101>

3.3.4.1.3 Sending Compounded Responses

The server MAY compound responses to the client. If the server is sending a response to a compounded series of client requests that were related operations, as specified in section 3.2.4.1.4, the server MUST compound the response.<102>

To compound responses, the server MUST set the **NextCommand** in the first response to the offset, in bytes, from the beginning of the SMB2 header of the first response to the beginning of the 8-byte aligned SMB2 header in the subsequent response. This process MUST be done for each response except the final response in the chain, whose **NextCommand** MUST be set to 0. Then the entire response chain MUST be sent to the client as a single submission to the underlying transport.

3.3.4.2 Sending an Interim Response for an Asynchronous Operation

The server MAY choose to send an interim response for any request that is received. It MUST send an interim response for any request that could potentially block for an indefinite amount of time (such as pipe operations, directory change notifications, blocking byte-range-lock operations, and creates blocked by an oplock break). If an operation would require asynchronous processing but resources are constrained, the server MAY <103> choose to fail that operation with STATUS_INSUFFICIENT_RESOURCES. Otherwise the server MUST send an interim response. An interim response indicates to the client that the request has been received and a full response will come later.

To send an interim response for a request, the server MUST generate an asynchronous identifier for it by some method, as specified in section 3.3.1.1. This identifier MUST be unique for each asynchronous request currently active on this transport connection. The server MUST insert the request in **Connection.AsyncCommandList**.

The server MUST construct a response packet for the request. The SMB2 header of the response MUST be identical to that in the request with the following changes:

- It MUST set the **Status** field in the SMB2 header to STATUS_PENDING.
- The **NextCommand** field MUST be set to 0. If this response is later combined with other responses into a compounded response, as specified in section 3.3.4.1.3, this value will change later. The server SHOULD fail requests in a compound chain that try to go async. <104>
- The server MUST set the SMB2_FLAGS_SERVER_TO_REDIR bit in the **Flags** field of the SMB2 header.
- The server MUST set the SMB2_FLAGS_ASYNC_COMMAND bit in the **Flags** field of the SMB2 header.
- It MUST set the **AsyncId** field of the SMB2 header to the value that was generated earlier.

It MUST append an SMB2 ERROR response following the SMB2 header, as specified in section 2.2.2, with a **ByteCount** of zero. It MUST locate the request in **Connection.RequestList** and set its state to Asynchronous. This response MUST be sent to the client.

When the operation completes, the server MUST set the **AsyncId** field of the final response to the identifier generated and returned to the client in the asynchronous response. It MUST also set the SMB2_FLAGS_ASYNC_COMMAND bit in the **Flags** field of the SMB2 header. All other fields of the final response MUST be identical to what is described for the individual operations. The server MUST remove the request from **Connection.AsyncCommandList**.

3.3.4.3 Sending an Error Response

When the server is responding with a failure to any command sent by the client, the response message MUST be constructed as described here. An error code other than one of the following indicates a failure:

- STATUS_SUCCESS indicating that the operation completed successfully.
- STATUS_MORE_PROCESSING_REQUIRED in a SESSION_SETUP Response specified in section 2.2.6.
- STATUS_BUFFER_OVERFLOW in a QUERY_INFO Response specified in section 2.2.38.
- STATUS_BUFFER_OVERFLOW in a FSCTL_PIPE_TRANSCEIVE, FSCTL_PIPE_PEEK or FSCTL_DFS_GET_REFERRALS Response specified in section 2.2.32.
- STATUS_BUFFER_OVERFLOW in a READ Response on named pipe specified in section 2.2.20.
- STATUS_INVALID_PARAMETER in an IOCTL Response returning SRV_COPYCHUNK_RESPONSE as described in section 2.2.32.1

The server MUST provide the error code of the failure and a data buffer to be returned with the error. If nothing is specified, the buffer MUST be considered to be zero bytes in length.

The server MUST fill in the SMB2 header of the error response to match the SMB2 header of the request with the following changes:

- The **Status** field of the SMB2 header MUST be set to the error code provided.
- The **NextCommand** field MUST be set to 0. If this response is later combined with other responses into a compounded response, as specified in section 3.3.4.1.3, this value will change later.
- The SMB2_FLAGS_SERVER_TO_REDIR bit MUST be set in the **Flags** field of the SMB2 header.
- If the request is being handled asynchronously, the server MUST set the **AsyncId** field to the asynchronous identifier generated for this request, and set the SMB2_FLAGS_ASYNC_COMMAND bit in the **Flags** field.

Following the SMB2 header MUST be an SMB2 error response structure, as specified in section 2.2.2. The **ByteCount** of this response MUST be set to the length of the buffer that is provided as part of the error. If **ByteCount** is greater than zero, the server MUST place the data given in the error buffer in the **ErrorData** array of the response. This response MUST then be sent to the client.

3.3.4.4 Server Application Requests Session Key

An application running on the server issues a query for a user session key, specifying the name of the client computer and the security principal for the user. The application also provides a 16-byte buffer to receive the session key.

The server MUST walk through the entries in the GlobalSessionTable and locate the session for this security principal where Session.Connection indicates a connection from the client machine. If a session is found that matches, the server MUST copy the **Session.SessionKey** into the calling application's output buffer and complete the call with success. If no session is found, the server MUST fail the call with STATUS_OBJECT_NAME_NOT_FOUND.

3.3.4.5 Object Store Indicates an Oplock Break

The file system on the local resource indicates the breaking of an opportunistic lock, specifying the LocalOpen and the new oplock level, and expects the new oplock level in return. The new oplock level MUST be either SMB2_OPLOCK_LEVEL_NONE or SMB2_OPLOCK_LEVEL_II.

The server MUST locate the open by walking the GlobalOpenTable to find an entry whose **Open.LocalOpen** matches the one provided in the oplock break. If no entry is found, the break indication MUST be ignored and the server MUST complete the oplock break call with SMB2_OPLOCK_LEVEL_NONE as the new oplock level.

If an entry is found, the server MUST check the state of **Open.Connection**. If **Open.Connection** is zero, the server MUST remove the Open from the GlobalOpenTable, close the **Open.LocalOpen**, and free the open. The server MUST then complete the oplock break call with SMB2_OPLOCK_LEVEL_NONE as the new oplock level.

If **Open.Connection** is not zero, the server MUST construct an SMB2 OPLOCK_BREAK notification message following the syntax specified in section 2.2.23 to send back to the client. The server MUST set the Command in the SMB2 header to SMB2 OPLOCK_BREAK and the MessageId to 0xFFFFFFFFFFFFFFFF. The **FileId** field of the response structure MUST be set to the values from the Open structure, with the volatile part set to **Open.FileId** and the persistent part set to **Open.DurableFileId**. The oplock Level of the response MUST be set to the value provided by the object store. The server MUST set **Open.OplockState** to Breaking. This response is sent to the client. This response MUST NOT be signed, as specified in section 3.3.4.1.1. The server MUST start the oplock break acknowledgment timer as specified in section 3.3.2.1.

3.3.5 Message Processing Events and Sequencing Rules

The SMB 2.0 Protocol server is driven by a series of request messages sent by the client. Processing for these messages is determined by the command in the SMB2 header of the response and is detailed for each of the SMB2 response messages below.

3.3.5.1 Accepting an Incoming Connection

When the server accepts an incoming connection from any of its registered transports, it MUST allocate a Connection object for it. The Connection object is initialized as described here.

Connection.SessionTable is set to an empty table.

Connection.CommandSequenceWindow is set to a sequence window, as specified in section 3.3.1.2, with a starting receive sequence of 0 and a window size of 1.

Connection.AsyncCommandList is set to an empty list.

Connection.RequestList is set to an empty list.

Connection.ClientCapabilities is set to 0.

Connection.NegotiateReceived is set to FALSE.

This connection MUST be inserted into the global ConnectionList.

3.3.5.2 Receiving Any Message

The incoming request is added to the **Connection.RequestList** before verifying the connection state, sequence number, or signature.

3.3.5.2.1 Verifying the Connection State

If the request being received is not an SMB2 NEGOTIATE request or a traditional SMB_COM_NEGOTIATE, as specified in section 1.7, and Connection.NegotiateReceived is FALSE, the server MUST disconnect the connection and send no reply.

3.3.5.2.2 Verifying the Sequence Number

The server MUST check the MessageId for every received request whose Command is not SMB2 CANCEL and make sure it is a valid value that falls within the **Connection.CommandSequenceWindow**, as specified in section 3.3.1.7. If the received request is an SMB_COM_NEGOTIATE, as specified in section 1.7, the server MUST assume that MessageId is zero for this request.

If the server determines that the MessageId for the incoming request is not valid, based on the **Connection.CommandSequenceWindow**, the server SHOULD fail the request, as specified in section 3.3.4.3, with the error code STATUS_INVALID_PARAMETER, and SHOULD disconnect the connection.<105>

Otherwise, it MUST remove this MessageId from the **Connection.CommandSequenceWindow**.

3.3.5.2.3 Verifying the Signature

If Session.ShouldSign is FALSE, no verification is necessary.

If the SMB2 header of the request has SMB2_FLAGS_SIGNED set in the **Flags** field, the server MUST verify the signature. The server MUST look up the session in the **Connection.SessionTable** using the SessionId in the SMB2 header of the request. If the session is not found, the request MUST be failed, as specified in section Sending an Error Response (section 3.3.4.3), with the error code STATUS_USER_SESSION_DELETED. If the session is found, the server MUST verify the signature of the message as specified in section 3.1.5.1, using **Session.SessionKey** as the session key, and passing in the request message. If the signature verification fails, the server MUST fail the request with the error code STATUS_ACCESS_DENIED. The server MAY also choose to disconnect the connection. If signature verification succeeds, the server MUST continue processing on the packet.<106>

If the SMB2 header of the request does not have SMB2_FLAGS_SIGNED set in the **Flags** field, the server MUST determine if the client failed to sign a packet that required it. The server MUST look up the session in the **Connection.SessionTable** using the SessionId in the SMB2 header of the request. If the session is found and **Session.ShouldSign** is equal to TRUE, the server MUST fail this request with STATUS_ACCESS_DENIED. The server MAY also choose to disconnect the connection. If either the session is not found, or **Session.ShouldSign** is FALSE, the server continues processing on the packet.<107>

3.3.5.2.4 Handling Incorrectly Formatted Requests

If the server receives a request that does not conform to the structures outlined in section 2, the server MUST fail the request, as specified in section 3.3.4.3, with the error code STATUS_INVALID_PARAMETER. The server MAY also disconnect the connection.<108>

The server MUST disconnect without sending an error response if any of the following are true:

- The ProtocolId field in the SMB2 Header is not equal to 0xFE, 'S', 'M', and 'B' (in network order).
- The Command code in the SMB2 header does not match one of the command codes in the SMB2 header as specified in section 2.2.1.

- The server receives a request with a length less than the length of the SMB2 header as specified in section 2.2.1.

3.3.5.2.5 Handling Compounded Requests

If the **NextCommand** field in the SMB2 header of the request is not equal to 0, the server MUST process the received request as a compounded series of requests. The server SHOULD fail requests in a compound chain that try to go async.<109> There are two different styles of compounded requests, which are described in the following sections.

3.3.5.2.5.1 Handling Compounded Unrelated Requests

If SMB2_FLAGS_RELATED_OPERATIONS is not set in the **Flags** field of the SMB2 header of every request except the first one, the receive MUST be handled as a series of compounded unrelated requests.

The server MUST handle each individual request described in the chain separately. The length of each request is determined by the **NextCommand** value in the SMB2 header of the request. The length of the final request is equal to the length between the beginning of SMB2 header and the end of the received buffer.

3.3.5.2.5.2 Handling Compounded Related Requests

If SMB2_FLAGS_RELATED_OPERATIONS is set in the **Flags** field of the SMB2 header of all requests except the first one, the receive MUST be handled as a series of compounded related requests. If the first request has SMB2_FLAGS_RELATED_OPERATIONS set the server MUST fail processing the compound chain request.

The server MUST handle each individual request that is described in the chain in order. For the first request, the identifiers for **FileId**, **SessionId**, and **TreeId** MUST be taken from the received request. For every subsequent request, the values for **FileId**, **SessionId**, and **TreeId** MUST be the ones used in previous request or generated from the previous resulting response. When all operations are complete, the responses MUST be compounded into a single response to return to the client.

3.3.5.3 Receiving an SMB_COM_NEGOTIATE

If the request does not have a valid SMB2 header following the syntax specified in section 2.2.1, the server MUST check to see if it has received an SMB_COM_NEGOTIATE as specified in section 1.7. If **Connection.NegotiateReceived** is TRUE, the server MUST fail this request with STATUS_INVALID_PARAMETER.

This request is defined in [MS-SMB] section 2.2.2, with the SMB header defined in section 2.2.1. If the request matches the format described there, the server MUST scan the dialects provided for the dialect string "SMB 2.002".<110> If the string is present, the client understands SMB2, and the server MUST respond with an SMB2 NEGOTIATE response. If the string is not present in the dialect list and the server also implements SMB as specified in [MS-SMB], it MUST terminate SMB2 processing on this connection and start SMB processing on this connection. If the string is not present in the dialect list and the server does not implement SMB, the server MUST disconnect the connection without sending a response.

The server MUST set the command of the SMB2 header to SMB2 NEGOTIATE. All other values MUST be set following the syntax specified in section 2.2.1, and any value not defined there with a default MUST be set to 0. The header is followed by an SMB2 NEGOTIATE response that MUST be constructed as specified in section 2.2.4, with the following specific values:

- SecurityMode MUST have the SMB2_NEGOTIATE_SIGNING_ENABLED bit set.
- If RequireMessageSigning is TRUE, the server MUST also set SMB2_NEGOTIATE_SIGNING_REQUIRED in the SecurityMode.
- DialectRevision MUST be set to 0x0202.<111>
- ServerGuid is set to the global ServerGuid value.
- Capabilities is set to SMB2_GLOBAL_CAP_DFS.
- MaxTransactSize is set to the maximum buffer size, in bytes, that this server allows on the transport that established this connection for QUERY_INFO, QUERY_DIRECTORY, SET_INFO and CHANGE_NOTIFY operations. This field is applicable only for buffers sent by the client in SET_INFO requests, or returned from the server in QUERY_INFO, QUERY_DIRECTORY and CHANGE_NOTIFY responses. <112>
- MaxReadSize is set to the maximum size, in bytes, of the Length in an SMB2 READ Request (section 2.2.19) that the server will accept on the transport that established this connection.
- MaxWriteSize is set to the maximum size, in bytes, of the Length in an SMB2 READ Request (section 2.2.19) that the server will accept on the transport that established this connection.
- SystemTime is set to the current time, in FILETIME format as specified in [MS-DTYP] section 2.3.1.
- ServerStartTime is set to the global ServerStartTime value.
- SecurityBufferOffset is set to the offset to the Buffer[] field in the response in bytes from the beginning of the SMB2 header.
- SecurityBufferLength is set to the length of the GSS token being returned in the negotiate response, generated as described below.
- Buffer[] is filled with the GSS token being returned in the negotiate response as described below.

The generation of the GSS token for SMB2 negotiate response MUST be done as specified in [MS-SPNG] section 3.2.5.1. The server MUST initialize the mechanism with the Integrity, Confidentiality, and Delegate options and use the server-initiated variation as specified in [MS-SPNG] section 3.2.5.1.

Connection.NegotiateReceived MUST be set to TRUE, and the response is sent to the client.

3.3.5.4 Receiving an SMB2 NEGOTIATE Request

When the server receives a request with an SMB2 header with a Command value equal to SMB2 NEGOTIATE, it MUST process it as follows:

If Connection.NegotiateReceived is TRUE, the server MUST fail the request with STATUS_INVALID_PARAMETER.<113>

The server checks to see if the dialect number 0x0202<114> was provided in the Dialect array of the SMB2_REQ_NEGOTIATE request. If it is not found, the server MUST fail the request with STATUS_NOT_SUPPORTED.

If it is found, the server MUST construct a negotiate response as specified in section 3.3.5.3. Connection.NegotiateReceived MUST be set to TRUE, and the response MUST be sent to the client with STATUS_SUCCESS.

3.3.5.5 Receiving an SMB2 SESSION_SETUP Request

When the server receives a request with an SMB2 header with a Command value equal to SMB2 SESSION_SETUP, message handling proceeds as follows:

1. If **SessionId** in the SMB2 header of the request is zero, the server MUST process the authentication request as specified in section 3.3.5.5.1.
2. The server MUST look up the session in the **Connection.SessionTable** using the **SessionId** from the SMB2 header.
3. If the session is not found, the server MUST fail the session setup request with STATUS_USER_SESSION_DELETED. Otherwise, proceed to step 4.
4. If **Session.State** is Expired, the server MUST process the session setup request as specified in section 3.3.5.5.2. Otherwise, proceed to step 5.
5. If **Session.State** is Valid, the server MUST fail the session setup request with STATUS_REQUEST_NOT_ACCEPTED. Otherwise, proceed to step 6.
6. The server MUST continue processing the request as specified in section 3.3.5.5.3.

3.3.5.5.1 Authenticating a New Session

A session object MUST be allocated for this request. The session MUST be inserted into SessionTable, and the lookup identifier received from the table MUST be stored as **Session.SessionId**. The SMB2 server MUST reserve -1 for invalid SessionID. The other values MUST be initialized as follows:

- **Session.Connection** is set to the connection on which the request was received.
- **Session.State** is set to InProgress.
- **Session.SecurityContext** is set to NULL.
- **Session.SessionKey** is set to all zeros.
- **Session.ShouldSign** is set to FALSE.
- **Session.OpenTable** is set to an empty table.
- **Session.TreeConnectTable** is set to an empty table.

Using this session, authentication is continued as specified in section 3.3.5.5.3.

3.3.5.5.2 Reauthenticating an Existing Session

Session.State MUST be set to InProgress, and Session.SecurityContext set to NULL. Authentication is continued as specified in section 3.3.5.5.3, except that the server MUST NOT regenerate Session.SessionKey.

3.3.5.5.3 Handling GSS-API Authentication

The server MUST extract the GSS token from the request. The token is SecurityBufferLength bytes in length, and located SecurityBufferOffset bytes from the beginning of the SMB2 header. The server MUST use the configured GSS authentication protocol to obtain the next GSS output token for the authentication exchange. The input GSS token can be 0 bytes in length.

If the GSS mechanism indicates an error, the server MUST fail the session setup request with the error received from the GSS API call by placing the 32-bit NTSTATUS code received from the GSS API into the **Status** field of the SMB2 header. The session object MUST be removed from the **Connection.SessionTable**. Any opens in **Session.OpenTable** MUST be closed and freed. Any tree connects in **Session.TreeConnectTable** MUST be freed. The session object MUST also be freed, and the error response MUST be sent to the client.

The following errors can be returned by the GSS-API interface. A detailed description of these errors is specified in [MS-ERREF].

- STATUS_DOWNGRADE_DETECTED
- STATUS_NO_SUCH_LOGON_SESSION
- SEC_E_WRONG_PRINCIPAL
- STATUS_NO_SUCH_USER
- STATUS_ACCOUNT_DISABLED
- STATUS_ACCOUNT_RESTRICTION
- STATUS_ACCOUNT_LOCKED_OUT
- STATUS_WRONG_PASSWORD
- STATUS_SMARTCARD_WRONG_PIN
- STATUS_ACCOUNT_EXPIRED
- STATUS_PASSWORD_EXPIRED
- STATUS_INVALID_LOGON_HOURS
- STATUS_INVALID_WORKSTATION
- STATUS_PASSWORD_MUST_CHANGE
- STATUS_LOGON_TYPE_NOT_GRANTED
- STATUS_PASSWORD_RESTRICTION
- STATUS_SMARTCARD_SILENT_CONTEXT
- STATUS_SMARTCARD_NO_CARD
- STATUS_SMARTCARD_CARD_BLOCKED
- STATUS_PKINIT_FAILURE
- STATUS_PKINIT_CLIENT_FAILURE
- STATUS_PKINIT_NAME_MISMATCH
- STATUS_NETLOGON_NOT_STARTED
- STATUS_DOMAIN_CONTROLLER_NOT_FOUND
- STATUS_NO_SUCH_DOMAIN

- STATUS_BAD_NETWORK_PATH
- STATUS_TRUST_FAILURE
- STATUS_TRUSTED_RELATIONSHIP_FAILURE
- STATUS_NETWORK_UNREACHABLE
- SEC_E_INVALID_TOKEN
- SEC_E_NO_AUTHENTICATING_AUTHORITY
- SEC_E_NO_CREDENTIALS
- STATUS_INTERNAL_ERROR
- STATUS_NO_MEMORY
- SEC_E_NOT_OWNER
- SEC_E_CERT_WRONG_USAGE
- SEC_E_SMARTCARD_LOGON_REQUIRED
- SEC_E_SHUTDOWN_IN_PROGRESS

If the GSS mechanism indicates success, the server MUST construct an SMB2 SESSION_SETUP response, specified in section 2.2.6, as described here:

- SMB2_FLAGS_SERVER_TO_REDIR MUST be set in the **Flags** field of the SMB2 header.
- The output token received from the GSS mechanism MUST be returned in the response. SecurityBufferLength MUST indicate the length of the output token, and SecurityBufferOffset MUST indicate its offset, in bytes, from the beginning of the SMB2 header.
- **Session.SessionId** MUST be placed in the **SessionId** field of the SMB2 header.

If the GSS mechanism indicates that this is the final message in the authentication exchange, the following additional steps MUST be taken:

- The status code in the SMB2 header of the response MUST be set to STATUS_SUCCESS.
- If **Connection.ClientCapabilities** is 0, the server MUST set **Connection.ClientCapabilities** to the capabilities received in the SMB2 SESSION_SETUP request.
- If the security principal is an anonymous user, the server MUST set the SMB2_SESSION_FLAG_IS_NULL flag in the SessionFlags of the SMB2 SESSION_SETUP response.
- If the security principal is a guest user, the server MUST set the SMB2_SESSION_FLAG_IS_GUEST in the SessionFlags of the SMB2 SESSION_SETUP response.
- If either SMB2_SESSION_FLAG_IS_GUEST or SMB2_SESSION_FLAG_IS_NULL was set in the SessionFlags, then **Session.ShouldSign** MUST be set to FALSE. Otherwise, if the SecurityMode of the client request has the SMB2_NEGOTIATE_SIGNING_REQUIRED bit set OR the global RequireMessageSigning is set to TRUE, **Session.ShouldSign** MUST be set to TRUE.
- If **Session.ShouldSign** is TRUE, the server MUST sign the final session setup response before sending it to the client.

- If **Session.SecurityContext** is NULL, it MUST be set to a value representing the user which successfully authenticated this connection. The security context MUST be obtained from the GSS authentication subsystem. If it is not NULL, no changes are necessary.
- If **Session.SessionKey** is uninitialized, the server MUST query the session key for this authentication from the underlying authentication protocol, and store the session key in **Session.SessionKey**. For how this is calculated for Kerberos authentication via GSS-API, see [MS-KILE] section 3.1.1.2. For how this is calculated for NTLM authentication via GSS-API, see [MS-NLMP] section 3.1.5.1.
- If the **PreviousSessionId** field of the request is not equal to zero, the server MUST take the following actions:
  - The server MUST look up the old session based on this value. If no session is found, no other processing is necessary.
  - If a session is found with **Session.SessionId** == PreviousSessionId, the server MUST determine if the old session and the newly established session are created by the same user by comparing the user identifiers obtained from the **Session.SecurityContext** on the new and old session.
    - If the server determines the authentications were for the same user, the server MUST remove the old session from the Session Table. Any opens in Session.OpenTable of the old session MUST be closed and freed. Any tree connects in Session.TreeConnectTable of the old session MUST be freed.
    - If the server determines that the authentications were for different users, the server MUST ignore the PreviousSessionId value.
- **Session.State** MUST be set to Valid.
- **Session.ExpirationTime** MUST be set to the expiration time returned by the GSS authentication subsystem. If the GSS authentication subsystem does not return an expiration time, the Session.ExpirationTime should be set to infinity.

The GSS-API can indicate that this is not the final message in authentication exchange using the GSS_S_CONTINUE_NEEDED semantics as specified in [MS-SPNG] section 3.1.1. If the GSS mechanism indicates that this is not the final message of the authentication exchange, the following additional steps MUST be taken:

- The status code in the SMB2 header of the response MUST be set to STATUS_MORE_PROCESSING_REQUIRED.

3.3.5.6 Receiving an SMB2 LOGOFF Request

When the server receives a request with an SMB2 header with a Command value equal to SMB2 LOGOFF, message handling MUST proceed as follows:

The server MUST locate the session being logged off, using the **SessionId** in the SMB2 header of the request to do a lookup on the SessionTable. If no session is found, or if **Session.Connection** is not equal to the connection on which the request was received, the server MUST fail the request with STATUS_USER_SESSION_DELETED.

Otherwise, the server MUST remove this session from the SessionTable. Any opens in **Session.OpenTable** of the old session MUST be closed and freed. Any tree connects in **Session.TreeConnectTable** of the old session MUST be freed. The server MUST construct an SMB2

LOGOFF response with a status code of STATUS_SUCCESS, following the syntax specified in section 2.2.6, and send it to the client. The session itself is then freed.

3.3.5.7 Receiving an SMB2 TREE_CONNECT Request

When the server receives a request with an SMB2 header with a Command value equal to SMB2 TREE_CONNECT, message handling proceeds as follows:

The server MUST locate the authenticated session that is attempting to connect to a share by looking up the session object in SessionTable using the SessionId provided in the SMB2 header of the request. If the session is not found, or if **Session.Connection** of the session found is not the same as the connection on which this request was received, the server MUST fail the request with STATUS_USER_SESSION_DELETED. If the session is found, but **Session.State** is Expired, the server MUST fail the request with STATUS_NETWORK_SESSION_EXPIRED. If the session is found, but **Session.State** is InProgress, the server MUST fail the request with STATUS_ACCESS_DENIED.

The server MUST locate the share being connected to by passing the path provided in the request to do a lookup in ShareList. If no share with a matching path is found, the server MUST fail the request with STATUS_BAD_NETWORK_NAME. If a share is found, the server MUST determine whether the user represented by **Session.SecurityContext** should be granted access based on the authorization policy specified in **Share.ConnectSecurity**. If the server determines that access should not be granted, the server MUST fail the request with STATUS_ACCESS_DENIED. Otherwise, the server MUST allocate a tree connect object and insert it into **Session.TreeConnectTable** with the following default values:

- **TreeConnect.TreeId** MUST be set to a value generated to uniquely identify this tree connect in the **Session.TreeConnectTable**. The SMB2 server MUST reserve -1 for invalid TreeID.
- **TreeConnect.Session** MUST be set to the session found on the **SessionId** lookup.
- **TreeConnect.Share** MUST be set to the share found on the lookup.

The tree connect response MUST be constructed following the syntax specified in section 2.2.12, as described here:

- ShareFlags MUST be set based on the individual share properties (**Share.CscFlags**, **Share.DoAccessBasedDirectoryEnumeration**, **Share.AllowNamespaceCaching**, **Share.ForceSharedDelete**, **Share.RestrictExclusiveOpens**.)
- ShareType MUST be set based on the resource being shared:
  - If this share provides access to named pipes, ShareType MUST be set to SMB2_SHARE_TYPE_PIPE.
  - If this share provides access to a printer, ShareType MUST be set to SMB2_SHARE_TYPE_PRINT.
  - Otherwise, ShareType MUST be set to SMB2_SHARE_TYPE_DISK.
- If **Share.IsDfs** is TRUE, Capabilities MUST be set to SMB2_SHARE_CAP_DFS. Otherwise, it MUST be set to 0.
- MaximalAccess MUST be set to the highest access the user described by **Session.SecurityContext** would have when accessing resources underneath the security descriptor **Share.FileSecurity**.

The response MUST then be sent to the client.

3.3.5.8 Receiving an SMB2 TREE_DISCONNECT Request

When the server receives a request with an SMB2 header having a Command value equal to SMB2 TREE_DISCONNECT, message handling proceeds as follows:

Session Verification:

The server MUST locate the session, using the **SessionId** in the SMB2 header of the request to do a lookup on the SessionTable. If no session is found, or if **Session.Connection** is not equal to the connection on which the request was received, the server MUST fail the request with STATUS_USER_SESSION_DELETED. If **Session.State** is set to Expired, the server MUST fail the request with STATUS_NETWORK_SESSION_EXPIRED. If **Session.State** is set to InProgress, the server MUST fail the request with STATUS_ACCESS_DENIED.

Tree Connect Verification:

The server MUST locate the tree connection being disconnected using the **TreeId** in the SMB2 header of the request to do a lookup in **Session.TreeConnectTable**. If no tree connect is found, the server MUST fail the request with STATUS_NETWORK_NAME_DELETED.

The server MUST enumerate all the opens in **Session.OpenTable**. For any open where **Open.TreeConnect** is equal to the tree connect being disconnected, the server MUST close the open and free the underlying structure.

The tree connect MUST then be removed from **Session.TreeConnectTable** and freed. The server MUST initialize an SMB2_TREE_DISCONNECT response following the syntax specified in section 2.2.12, and send it to the client.

3.3.5.9 Receiving an SMB2 CREATE Request

When the server receives a request with an SMB2 header with a Command value equal to SMB2 CREATE, message handling proceeds as described in the following sections.

Session Verification:

The server MUST locate the session performing the create, using the SessionId in the SMB2 header of the request to do a lookup on the SessionTable. If no session is found, or if **Session.Connection** is not equal to the connection on which the request was received, the server MUST fail the request with STATUS_USER_SESSION_DELETED. If **Session.State** is set to Expired, the server MUST fail the request with STATUS_NETWORK_SESSION_EXPIRED. If **Session.State** is set to InProgress, the server MUST fail the request with STATUS_ACCESS_DENIED.

Tree Connect Verification:

The server MUST locate the tree connection used in the create using the **TreeId** in the SMB2 header of the request to do a lookup in **Session.TreeConnectTable**. If no tree connect is found, the request MUST be failed with STATUS_NETWORK_NAME_DELETED.

Path Name Validation:

The server MUST extract the target path name for the create from the SMB2 CREATE request. The path name MUST be validated to ensure that it is NameLength bytes in length, and is at an offset of NameOffset bytes from the beginning of the SMB2 header. If the request received has SMB2_FLAGS_DFS_OPERATIONS set in the **Flags** field of the SMB2 header, and **TreeConnect.Share.IsDfs** is TRUE, the server MUST invoke DFS to normalize the path name (for more information, see [MSDFS]). If normalization fails, the server MUST fail the create request with the error code returned by the DFS normalization routine. Otherwise, the server MUST continue

processing. If the normalization procedure returns an altered target name, the modified name MUST be used for further operations.

If the share that is the target of the create request is a printer, the server MUST fail requests with STATUS_OBJECT_NAME_NOT_FOUND if the **CreateDisposition** value is FILE_OPEN or FILE_OVERWRITE. The server MUST only allow opens for write access (where DesiredAccess is FILE_WRITE_DATA, FILE_APPEND_DATA, or GENERIC_WRITE). Other values for **CreateDisposition** SHOULD NOT be set for a printer share, as specified in section 2.2.13 and the server MUST fail those requests with STATUS_NOT_SUPPORTED.

The description contained here is for a generic create operation. Sections 3.3.5.9.1 through 3.3.5.9.7 detail server behavior when various create contexts are provided in the request, and describe how that affects server operation.

Open Execution:

The server MUST attempt to open the named object in the underlying object store using the parameters specified for DesiredAccess, FileAttributes, ShareAccess, **CreateDisposition**, CreateOptions, SecurityFlags, ImpersonationLevel, and the PathName. The PathName MUST be parsed relative to **TreeConnect.Share.LocalPath**. The server MUST map these flags to match the semantics of its implementation-specific object store. See section 2.2.13 for more details on the exact meaning of the various flags and options. Section 2.2.13 also lists valid values and error codes for invalid values of these flags and options. If the underlying object store returns a failure for the attempted Open, the server MUST send an SMB2 error response with the appropriate error code as specified in section 2.2.2. The same rules apply when opening named pipe and print files, except that the underlying object store MAY choose not to support some of the flags and options. The flags and options that are not supported when opening named pipes and print files are specified in section 2.2.13.

Failed Open Handling:

If the underlying object store returns a failure indicating that the attempted open operation failed due to the presence of a symbolic link in the target path name, the server MUST fail the create operation with the error code STATUS_STOPPED_ON_SYMLINK, and pass back the error to the client by constructing an error response as specified in section 2.2.2.1.

Successful Open Initialization:

If the open is successful, the server MUST allocate an open object for this open and insert it into both **Session.OpenTable** and GlobalOpenTable. The initial values MUST be set as follows:

- **Open.FileId** is set to a generated value that uniquely identifies this Open in **Session.OpenTable**. The SMB2 server MUST reserve -1 for invalid FileID.
- **Open.DurableFileId** is set to a generated value that uniquely identifies this open in GlobalOpenTable.
- **Open.Session** is set to refer to the session that performed the open.
- **Open.Connection** is set to refer to the connection on which the open request was received.
- **Open.LocalOpen** is set to the open of the object in the local resource received as part of the local create operation.
- **Open.GrantedAccess** is the access granted to the caller for the open by the underlying object store. It MUST be equal to the DesiredAccess specified in the request, except in the case where MAXIMAL_ACCESS is included in the DesiredAccess.

- **Open.OplockLevel** is set to None.
- **Open.OplockState** is set to None.
- **Open.OplockTimeout** is set to 0.
- **Open.IsDurable** is set to FALSE.
- **Open.DurableOpenTimeout** is set to 0.
- **Open.DurableOwner** is set to NULL.
- **Open.EnumerationLocation** is set to 0.
- **Open.EnumerationSearchPattern** is set to an empty string.

Oplock Acquisition:

If the open is successful, the shared resource is not a named pipe, and the RequestedOplockLevel is not SMB2_OPLOCK_LEVEL_NONE, the server MUST attempt to acquire an opportunistic lock on the open from the underlying object store. If the underlying object store grants the oplock, then **Open.OplockState** is set to Held and **Open.OplockLevel** is set to the level of the oplock acquired.

Response Construction:

The server MUST construct a response following the syntax specified in section 2.2.14. The values MUST be set as follows:

- OplockLevel is set to **Open.OplockLevel**.
- CreateAction is set to the action taken by the create following the syntax specified in section 2.2.14.
- CreationTime is set to the value queried from the object store for when the object was created.
- LastAccessTime is set to the value queried from the object store for when the object was last accessed.
- LastWriteTime is set to the value queried from the object store for when the object was last written to.
- LastChangeTime is set to the value queried from the object store for when the object was last modified, including attribute changes.
- AllocationSize is set to the amount of space reserved for the object in bytes on the underlying store. If this is a named pipe, AllocationSize MUST be 0.
- EndOfFile is set to the size of the **main stream** of the object in bytes. For named pipes this value MUST be 0.
- FileAttributes MUST be set to the attributes of the object following the syntax specified in section 2.2.14.
- **FileId.Persistent** MUST be set to **Open.DurableFileId**.
- **FileId.Volatile** MUST be set to **Open.FileId**.
- CreateContextLength and CreateContextOffset MUST be set to 0.

This response MUST be sent back to the client.

The following create contexts are potentially received as part of the create request. In each subsection, handling this create context is outlined. Unless otherwise noted, any combination of the create contexts listed in the following sections is valid.

3.3.5.9.1 Handling the SMB2_CREATE_EA_BUFFER Create Context

The client is requesting that an array of extended attributes be applied to the file that is being created. This create context MAY be combined with any of those listed here except SMB2_CREATE_DURABLE_HANDLE_RECONNECT.

The processing changes involved for this create context are:

In the "Open Execution" phase, the server MUST pass the received extended attributes array to the underlying object store to be stored on the created file. If the object store does not support extended attributes, the server MUST fail the open request with STATUS_EAS_NOT_SUPPORTED.

3.3.5.9.2 Handling the SMB2_CREATE_SD_BUFFER Create Context

The client is requesting that a specific security descriptor be applied to the file that is being created. This create context can be combined with any of those listed here except SMB2_CREATE_DURABLE_HANDLE_RECONNECT.

The processing changes involved for this create context are:

In the "Open Execution" phase, the server MUST pass the received security descriptor to the underlying object store to be stored on the created file. If the object store does not support file security, the value MAY<115> be ignored or an appropriate error MAY be returned to the client.

3.3.5.9.3 Handling the SMB2_CREATE_ALLOCATION_SIZE Create Context

The client is requesting that a specific allocation size be set for the file that is being created. This create context can be combined with any of those listed here except SMB2_CREATE_DURABLE_HANDLE_RECONNECT. The server SHOULD support this create context request. <116> If the server does not support it, the SMB2_CREATE_ALLOCATION_SIZE create context request MUST be ignored.

The processing changes involved for this create context are:

In the "Open Execution" phase, the server MUST pass the received allocation size to the underlying object store to reserve the appropriate space for the created file. If the object store does not have sufficient space available to hold a file of the requested size, the server MUST fail the open request with STATUS_DISK_FULL.

3.3.5.9.4 Handling the SMB2_CREATE_TIMEWARP_TOKEN Create Context

The client is requesting that the create operation be performed on a **snapshot** of the underlying object store taken at a previous time. This create context can be combined with any of those listed here except SMB2_CREATE_DURABLE_HANDLE_RECONNECT.

The processing changes involved for this create context are:

In the "Path Name Validation" phase, the server MUST verify that a snapshot of the underlying object store at the time stamp provided in the create context exists. If it does not, the server MUST fail the request with STATUS_OBJECT_NAME_NOT_FOUND.

In the "Open Execution" phase, the server MUST perform the open on the snapshot of the underlying object store taken at the time specified, instead of using the current view of the object store.

3.3.5.9.5 Handling the SMB2_CREATE_QUERY_MAXIMAL_ACCESS Create Context

The client is requesting that the server return maximal access information for the open if the last modified time for the file, as returned by the underlying object store, is not equal to the time stamp provided by the client in the create context. This create context can be combined with any of those listed here except SMB2_CREATE_DURABLE_HANDLE_RECONNECT.

The processing changes involved for this create context are:

In the "Response Construction" phase, if the LastModifiedTime is not equal to the time received in the request create context, the server MUST calculate the maximal access that the user identified by **Session.SecurityContext** has on the object that was opened. It must construct a response create context, following the syntax specified in section 2.2.14.2.5, and include it in the buffer described by the response CreateContextLength and CreateContextOffset.

If the LastModifiedTime is equal to the time received in the request create context, then the server MUST NOT calculate or append the maximal access information to the response.

3.3.5.9.6 Handling the SMB2_CREATE_DURABLE_HANDLE_REQUEST Create Context

The client is requesting that the open be marked for durable operation. This create context can be combined with any of those listed here except SMB2_CREATE_DURABLE_HANDLE_RECONNECT.

The processing changes involved for this create context are:

In the "Successful Open Initialization" phase, the server MUST set **Open.IsDurable** to TRUE. This permits the client to use **Open.DurableFileId** to request a reopen of the file on a subsequent request as specified in section 3.3.5.9.7.

3.3.5.9.7 Handling the SMB2_CREATE_DURABLE_HANDLE_RECONNECT Create Context

The client is requesting a reconnect to an existing durable open. This create context MUST NOT be combined with any other create context.

There is no processing done for "Path Name Validation" or "Open Execution" as listed in the section above.

The processing changes involved for this create context are:

1. The server MUST look up an existing open in the GlobalOpenTable by doing a lookup with the **FileId.Persistent** portion of the create context.

2. If the lookup fails, the server MUST fail the request with STATUS_OBJECT_NAME_NOT_FOUND and proceed as specified in "Failed Open Handling" in section 3.3.5.9.

3. If **Open.IsDurable** is false, the server MUST fail the request with STATUS_OBJECT_NAME_NOT_FOUND and proceed as specified in "Failed Open Handling" in section 3.3.5.9.

4. If **Open.Connection** is present, the server MUST fail the request with STATUS_OBJECT_NAME_NOT_FOUND and proceed as specified in "Failed Open Handling" in section 3.3.5.9.

5. If the user represented by **Session.SecurityContext** is not the same user denoted by **Open.DurableOwner**, the server MUST fail the request with STATUS_FILE_CLOSED and proceed as specified in Failed Open Handling" in section 3.3.5.9.

6. The server MUST set the **Open.Connection** to refer to the connection that received this request.

7. The server MUST regenerate **Open.FileId.Volatile**.

8. The server MUST insert the open into the **Session.OpenTable** with the **FileId.Volatile** as the new key.

9. The "Successful Open Initialization" and "Oplock Acquisition" phases MUST be skipped, and processing MUST continue as specified in "Response Construction."

3.3.5.10 Receiving an SMB2 CLOSE Request

When the server receives a request with an SMB2 header with a Command value equal to SMB2 CLOSE, message handling proceeds as follows:

The server MUST locate the session that is taking the action, using the SessionId in the SMB2 header of the request to do a lookup on the SessionTable. If no session is found, or if **Session.Connection** is not equal to the connection on which the request was received, the server MUST fail the request with STATUS_USER_SESSION_DELETED.

Next, the server MUST locate the tree connect by performing a lookup in the **Session.TreeConnectTable**, using the **TreeId** in the SMB2 header of the request. If no tree connect is found, the server MUST fail the request with STATUS_NETWORK_NAME_DELETED.

Next, the server MUST locate the open being closed by performing a lookup in the **Session.OpenTable**, using **FileId.Volatile** of the request as the lookup key. If no open is found, or if **Open.DurableFileId** is not equal to **FileId.Persistent**, the server MUST fail the request with STATUS_FILE_CLOSED.

The server MUST remove the open from both **Session.OpenTable** and the GlobalOpenTable, close the underlying **Open.LocalOpen** and free the open object.

If SMB2_CLOSE_FLAG_POSTQUERY_ATTRIB is set in the **Flags** field of the request, the server MUST query the attributes of the file after the close. This gives the client the attributes that have been updated to take into account any cached writes or extends that may have happened. The attributes that MUST be queried are the creation time, last access time, last write time, change time, allocation size in bytes, end of file in bytes, and file attributes.

The server then MUST construct the response following the syntax specified in section 2.2.16. The values MUST be set as follows:

- If the attributes of the file were requested and can be fetched, the server MUST set the Flags field to SMB2_CLOSE_FLAG_POSTQUERY_ATTRIB. Otherwise Flags MUST be set to 0.
- If SMB2_CLOSE_FLAG_POSTQUERY_ATTRIB was set:
  - CreationTime, LastAccessTime, LastWriteTime, ChangeTime, AllocationSize, EndOfFile, and FileAttributes MUST be set to the values returned from the attribute query.

- If SMB2_CLOSE_FLAG_POSTQUERY_ATTRIB was not set:
  - CreationTime, LastAccessTime, LastWriteTime, ChangeTime, AllocationSize, EndOfFile, and FileAttributes MUST all be set to 0.

The response MUST then be sent to the client.

The Server MUST send an SMB2 CHANGE_NOTIFY Response with STATUS_NOTIFY_CLEANUP status code for all pending CHANGE_NOTIFY requests associated with the FileId that is closed.

3.3.5.11 Receiving an SMB2 FLUSH Request

When the server receives a request with an SMB2 header with a Command value equal to SMB2 FLUSH, message handling proceeds as follows:

The server MUST locate the session that is taking the action, using the SessionId in the SMB2 header of the request to do a lookup on the SessionTable. If no session is found, or if **Session.Connection** is not equal to the connection on which the request was received, the server MUST fail the request with STATUS_USER_SESSION_DELETED.

Next, the server MUST locate the tree connect by performing a lookup in the **Session.TreeConnectTable**, using the **TreeId** in the SMB2 header of the request. If no tree connect is found, the server MUST fail the request with STATUS_NETWORK_NAME_DELETED.

Next the server MUST locate the open being flushed by performing a lookup in the **Session.OpenTable**, using the **FileId.Volatile** of the request as the lookup key. If no open is found, or if **Open.DurableFileId** is not equal to **FileId.Persistent**, the server MUST fail the request with STATUS_FILE_CLOSED.

The server MUST issue a request to the underlying object store to flush any cached data for **Open.LocalOpen**. If this is a file, the object store MUST propagate any cached data to persistent storage. If this is a named pipe, the server MUST wait for all data written to the pipe to be consumed by a reader. This operation MUST block until the flush is complete. (The server SHOULD choose to handle this request asynchronously, as specified in section 3.3.4.2.)<117>

If the operation succeeds, the server MUST initialize a response following the syntax specified in section 2.2.18.

If the operation fails, the server MUST return the error code to the client.

3.3.5.12 Receiving an SMB2 READ Request

When the server receives a request with an SMB2 header with a Command value equal to SMB2 READ, message handling proceeds as follows:

The server MUST locate the session that is taking the action, using the SessionId in the SMB2 header of the request to do a lookup on the SessionTable. If no session is found, or if **Session.Connection** is not equal to the connection on which the request was received, the server MUST fail the request with STATUS_USER_SESSION_DELETED.

Next the server MUST locate the tree connect by performing a lookup in the **Session.TreeConnectTable**, using the **TreeId** in the SMB2 header of the request. If no tree connect is found, the server MUST fail the request with STATUS_NETWORK_NAME_DELETED.

Next the server MUST locate the open that is being read from, by performing a lookup in the **Session.OpenTable**, using the **FileId.Volatile** of the request as the lookup key. If no open is

found, or if **Open.DurableFileId** is not equal to FileId.Persistent, the server MUST fail the request with STATUS_FILE_CLOSED.

If **Open.GrantedAccess** does not allow for FILE_READ_DATA, the request MUST be failed with STATUS_ACCESS_DENIED.

The server MUST validate that the length to read is within its configured maximum read size. If not, it MUST fail the request with STATUS_INVALID_PARAMETER.

The server MUST return STATUS_FILE_LOCK_CONFLICT if the read is to a region that is exclusively locked by another open.

The server MUST issue a read against the underlying object store represented by **Open.LocalOpen** for the length, in bytes, given by Length, at the offset, in bytes, from the beginning of the file provided in Offset.

If the read is being executed on a pipe, the server MUST ignore the value of the Offset field.

If the read is being executed on a named pipe, and the pipe is in blocking mode (the default), the operation could block for a long time, so the server MAY be required to handle it asynchronously, as specified in section 3.3.4.2. To set a pipe into blocking mode, use an SMB2 SET_INFO request with the FilePipeInformation file information class, as specified in [MS-FSCC] section 2.4.28.<118>

If the read fails, the server MUST fail the request using the error code received from the read operation.

If the read succeeds, the server MUST construct a read response following the syntax specified in section 2.2.20 with the following values:

- DataLength MUST be set to the number of bytes read.
- DataRemaining MUST be set to 0.
- DataOffset MUST be set to the offset into the response in bytes from the beginning of the SMB2 header where the data is located. This is usually the offset, in bytes, to the Buffer [] field.
- The data MUST be copied into the response.

The response MUST then be sent to the client.

3.3.5.13 Receiving an SMB2 WRITE Request

When the server receives a request with an SMB2 header with a Command value equal to SMB2 WRITE, message handling proceeds as follows:

The server MUST locate the session that is taking the action, using the SessionId in the SMB2 header of the request to do a lookup on the SessionTable. If no session is found, or if **Session.Connection** is not equal to the connection on which the request was received, the server MUST fail the request with STATUS_USER_SESSION_DELETED.

Next the server MUST locate the tree connect by performing a lookup in the **Session.TreeConnectTable**, using the **TreeId** in the SMB2 header of the request. If no tree connect is found, the server MUST fail the request with STATUS_NETWORK_NAME_DELETED.

Next the server MUST locate the open being written to by performing a lookup in the **Session.OpenTable**, using **FileId.Volatile** of the request as the lookup key. If no open is found, or

if **Open.DurableFileId** is not equal to FileId.Persistent, the server MUST fail the request with STATUS_FILE_CLOSED.

If the range being written to is within the existing file size and Open.GrantedAccess does not include FILE_WRITE_DATA, or if the range being written to extends the file size and Open.GrantedAccess does not include FILE_APPEND_DATA, the server MUST fail the request with STATUS_ACCESS_DENIED.

The server MUST validate that the length to write is within its configured maximum write size. If not, it MUST fail the request with STATUS_INVALID_PARAMETER.

The server MUST issue a write to the underlying object store represented by **Open.LocalOpen** for the length, in bytes, given by Length, at the offset, in bytes, from the beginning of the file provided in Offset.

The server MUST return STATUS_FILE_LOCK_CONFLICT if the write is to a region that is locked by another open.

If the write fails, the server MUST fail the request with the error code received from the write.

If the write succeeds, the server MUST construct a write response following the syntax specified in section 2.2.22 with the following values:

- Count MUST be set to the bytes written.
- Remaining MUST be set to 0.
- WriteChannelInfoOffset MUST be set to 0.
- WriteChannelInfoLength MUST be set to 0.

The response MUST then be sent to the client.

3.3.5.14 Receiving an SMB2 LOCK Request

When the server receives a request with an SMB2 header with a Command value equal to SMB2 LOCK, message handling proceeds as follows:

The server MUST locate the session that is taking the action, using the SessionId in the SMB2 header of the request to do a lookup on the SessionTable. If no session is found, or if **Session.Connection** is not equal to the connection on which the request was received, the server MUST fail the request with STATUS_USER_SESSION_DELETED.

Next, the server MUST locate the tree connect by performing a lookup in the **Session.TreeConnectTable**, using the **TreeId** in the SMB2 header of the request. If no tree connect is found, the server MUST fail the request with STATUS_NETWORK_NAME_DELETED.

Next, the server MUST locate the open on which the client is requesting a lock or unlock by performing a lookup in the **Session.OpenTable**, using the **FileId.Volatile** of the request as the lookup key. If no open is found, or if **Open.DurableFileId** is not equal to **FileId.Persistent**, the server MUST fail the request with STATUS_FILE_CLOSED.

If the flags of the initial SMB2_LOCK in the Locks array of the request has SMB2_LOCKFLAG_UNLOCK set, the server MUST process the lock array as a series of unlocks. Otherwise, it MUST process the lock array as a series of lock requests.

3.3.5.14.1 Processing Unlocks

For each SMB2_LOCK entry in the Locks array, if either SMB2_LOCKFLAG_SHARED_LOCK or SMB2_LOCKFLAG_EXCLUSIVE_LOCK is set, the server MUST fail the request with STATUS_INVALID_PARAMETER and stop processing further entries in the Locks array.

For each SMB2_LOCK entry in the Locks array, if SMB2_LOCKFLAG_FAIL_IMMEDIATELY is set, the server MAY ignore this value and continue processing next entry in the Locks array. <119>

The server MUST issue the byte-range unlock request to the underlying object store using **Open.LocalOpen**, and passing the Offset and Length (in bytes) from the SMB2_LOCK entry. If the unlock operation fails, the server MUST fail the operation with the error code received from the object store and stop processing further entries in the Locks array.

If the unlock operation succeeds and there are remaining entries in the Locks array, the server MUST continue processing the next entry in the Locks array as specified above.

If the unlock operation succeeds and there are no remaining entries in the Locks array, the server initializes an SMB2_LOCK response following the syntax specified in section 2.2.27, which then MUST be sent to the client.

3.3.5.14.2 Processing Locks

For combinations of Lock Flags other than those that are defined in the Flags field of section 2.2.26.1, the server SHOULD fail the request with STATUS_INVALID_PARAMETER<120>

The server MUST issue a byte-range lock request to the underlying object store using **Open.LocalOpen** and passing the Offset and Length (in bytes) from the SMB2_LOCK entry. If SMB2_LOCKFLAG_SHARED_LOCK is set, the lock MUST be acquired in a manner that allows read operations and other shared lock operations from other opens, but disallows writes to the region specified by the lock. If SMB2_LOCKFLAG_EXCLUSIVE_LOCK is set, the lock MUST be acquired in a manner that does not allow read, write, or lock operations from other opens for the range specified.

If the range being locked is already locked by another open in a way that does not allow this open to take a lock on the range, and if SMB2_LOCKFLAG_FAIL_IMMEDIATELY is set, the server MUST fail the request with STATUS_LOCK_NOT_GRANTED. Otherwise, the server MUST wait for the range in question to become available before completing the request.

If the lock operation fails, the server MUST unlock any ranges locked as part of processing the previous entries in the Lock element array of this request. It MUST stop processing any remaining entries in the Lock element array, and MUST fail the operation with the error code received from the lock operation.

If the lock operation succeeds and there are remaining entries in the Locks array, the server MUST continue processing the next entry in the Locks array as described above.

If the lock operation succeeds and there are no remaining entries in the Locks array, the server MUST construct an SMB2_RESP_LOCK response following the syntax specified in section 2.2.27, which is then sent to the client.

3.3.5.15 Receiving an SMB2 IOCTL Request

When the server receives a request with an SMB2 header with a Command value equal to SMB2 IOCTL, message handling proceeds as follows:

The server MUST locate the session that is taking the action, using the **SessionId** in the SMB2 header of the request to do a lookup on the SessionTable. If no session is found, or if Session.Connection is not equal to the connection on which the request was received, the server MUST fail the request with STATUS_USER_SESSION_DELETED.

Next, the server MUST locate the tree connect by performing a lookup in the **Session.TreeConnectTable**, using the **TreeId** in the SMB2 header of the request. If no tree connect is found, the server MUST fail the request with STATUS_NETWORK_NAME_DELETED.

If the **CtlCode** is either FSCTL_DFS_GET_REFERRALS or FSCTL_PIPE_WAIT and **FileID** in the SMB2 header of the request is not -1, then the server MUST fail the request with STATUS_INVALID_PARAMETER.

For **CtlCode** values other than FSCTL_DFS_GET_REFERRALS and FSCTL_PIPE_WAIT, the server MUST locate the open on which the client is requesting the operation by performing a lookup in Session.OpenTable using the **FileId.Volatile** field of the request as the lookup key. If no open is found or if **Open.DurableFileId** is not equal to **FileId.Persistent**, the server MUST fail the request with STATUS_FILE_CLOSED.

Note that any padding that inserted in the message between the input buffer and output buffer, to align the output buffer to an 8-byte boundary if necessary, is not included in the size of either the input or the output buffer.

Processing for a specific **CtlCode** is as specified in subsequent sections.

3.3.5.15.1 Handling an Enumeration of Previous Versions Request

When the server receives a request with an SMB2 header with a Command value equal to SMB2 IOCTL, and a **CtlCode** of FSCTL_SRV_ENUMERATE_SNAPSHOTS, message handling proceeds as follows:

The server MUST query the time stamps of available previous versions of the volume on which the file resides from the object store. If there are no previous versions available, the server MUST construct a SRV_SNAPSHOT_ARRAY following the syntax specified in section 2.2.32.2, setting NumberOfSnapShots to 0, NumberOfSnapShotsReturned to 0, and SnapShotArraySize to 0.

If there are previous versions available, the server MUST calculate the size required to return the previous version array. If the size of array required is larger than the maximum output buffer size received in the SMB2 IOCTL request, the server MUST construct a SRV_SNAPSHOT_ARRAY following the syntax specified in section 2.2.32.2 with NumberOfSnapShots set to the number of previous versions available, NumberOfSnapShotsReturned to 0, and SnapShotArraySize to the size required to receive the array of previous versions. If the maximum output buffer size specified is too small to contain this SRV_SNAPSHOT_ARRAY, the server must return the status STATUS_BUFFER_TOO_SMALL.

If the size of array required is less than or equal to the maximum output buffer size received in the SMB2 IOCTL request, the client MUST construct a SRV_SNAPSHOT_ARRAY with NumberOfSnapShots set to the number of previous versions available, NumberOfSnapShotsReturned set to the number of previous version time stamps being returned in the buffer, and SnapShotArraySize set to the size, in bytes, of the SnapShots buffer, which MUST then be filled in to hold the time stamps in textual GMT format for the previous version time stamps.

The server MUST then construct an SMB2 IOCTL response following the syntax specified in section 2.2.32, with the following values:

- **CtlCode** MUST be set to FSCTL_SRV_ENUMERATE_SNAPSHOTS.

- FileId.Persistent MUST be set to Open.DurableFileId. FileId.Volatile MUST be set to Open.FileId.
- InputOffset SHOULD be set to zero, and the receiver MUST ignore it. <121>
- InputCount MUST be set to 0.
- OutputOffset MUST be set to InputOffset + InputCount, rounded up to a multiple of 8.
- OutputCount MUST be set to the size of the SRV_SNAPSHOT_ARRAY that is constructed, as specified above.
- Flags MUST be set to 0.
- The server MUST copy the constructed SRV_SNAPSHOT_ARRAY into the Buffer field at the OutputOffset computed above.

The response MUST be sent to the client.

3.3.5.15.2 Handling a DFS Referral Information Request

When the server receives a request with an SMB2 header with a Command value equal to SMB2 IOCTL, and a **CtlCode** of FSCTL_DFS_GET_REFERRALS, message handling proceeds as follows:

The server MUST pass the MaxReferralLevel and the target name in the SMB2_REQ_GET_DFS_REFERRAL structure, as specified in [MS-DFSC] section 2.2.2, that was received in the request, as described by InputOffset and InputLength, to DFS, as specified in [MS-DFSC] section 2.2.2.

If DFS returns a failure, the server MUST fail the request with the error code received from DFS.

If DFS returns success and a response buffer containing the referrals, the server MUST then construct an SMB2 IOCTL response following the syntax specified in section 2.2.32, with the following values:

- **CtlCode** MUST be set to FSCTL_DFS_GET_REFERRALS.
- FileId MUST be set to { 0xFFFFFFFFFFFFFFFF, 0xFFFFFFFFFFFFFFFF }.
- InputOffset SHOULD be set to zero, and the receiver MUST ignore it. <122>
- InputCount MUST be set to 0.
- OutputOffset MUST be set to InputOffset + InputCount, rounded up to a multiple of 8.
- OutputCount MUST be set to the number of bytes received from DFS.
- Flags MUST be set to 0.
- The server MUST copy the buffer that was received from DFS into the Buffer field at the OutputOffset computed above.

The response MUST be sent to the client.

3.3.5.15.3 Handling a Pipe Transaction Request

When the server receives a request with an SMB2 header with a Command value equal to SMB2 IOCTL, and a **CtlCode** of FSCTL_PIPE_TRANSCEIVE, message handling proceeds as follows:

The server MUST attempt to write the number of bytes specified in the request by InputCount into the named pipe. If the write attempt fails, the server MUST fail the request returning the error code received from the named pipe.

The server MUST then attempt to read the number of bytes specified in the request by MaxOutputResponse from the named pipe. If the read attempt fails, the server MUST fail the request returning the error code received from the named pipe.

If the read attempt succeeds, the server MUST then construct an SMB2 IOCTL response following the syntax specified in section 2.2.32, with the following values:

- **CtlCode** MUST be set to FSCTL_PIPE_TRANSCEIVE.
- FileId.Persistent MUST be set to Open.DurableFileId. FileId.Volatile MUST be set to Open.FileId.
- InputOffset SHOULD be set to zero, and the receiver MUST ignore it. <123>
- InputCount SHOULD be set to 0.
- OutputOffset MUST be set to InputOffset + InputCount, rounded up to a multiple of 8.
- OutputCount MUST be set to the number of bytes read from the pipe.
- Flags MUST be set to 0.
- The server MUST copy the bytes read into the Buffer field at the OutputOffset computed above.

The response MUST be sent to the client.<124>

3.3.5.15.4 Handling a Peek at Pipe Data Request

When the server receives a request with an SMB2 header with a Command value equal to SMB2 IOCTL, and a **CtlCode** of FSCTL_PIPE_PEEK, message handling proceeds as follows:

The server MUST attempt to read the number of bytes specified in the request by MaxOutputResponse from the named pipe without removing the bytes from the pipe. If the read attempt fails, the server MUST fail the request and return the error code received from the named pipe. An FSCTL_PIPE_PEEK MUST never block.

If the read attempt succeeds, the server MUST then construct an SMB2 IOCTL response by following the syntax specified in section 2.2.32, with the following values:

- **CtlCode** MUST be set to FSCTL_PIPE_PEEK.
- FileId.Persistent MUST be set to Open.DurableFileId. FileId.Volatile MUST be set to Open.FileId.
- InputOffset SHOULD be set to zero, and the receiver MUST ignore it. <125>
- InputCount MUST be set to 0.
- OutputOffset MUST be set to InputOffset + InputCount, rounded up to a multiple of 8.
- OutputCount MUST be set to the number of bytes read from the pipe.
- Flags MUST be set to 0.
- The server MUST copy the bytes read into the Buffer field at the OutputOffset computed above.

The response MUST be sent to the client.

3.3.5.15.5 Handling a Source File Key Request

When the server receives a request with an SMB2 header with a Command value equal to SMB2 IOCTL, and a **CtlCode** of FSCTL_SRV_REQUEST_RESUME_KEY, message handling proceeds as follows.

The SRV_REQUEST_RESUME_KEY response is an opaque 24 byte blob followed by optional context as described in 2.2.32.3. <126>

The server MUST provide a 24-byte value that is used to uniquely identify the Open. The server SHOULD use Open.DurableFileId, or alternately, MAY use an internally generated value that is unique for all opens on the server. <127>

If the maximum output buffer size specified is too small to contain sn SRV_REQUEST_RESUME_KEY structure, the server MUST return the status STATUS_BUFFER_TOO_SMALL.

The server MUST construct an SMB2 IOCTL response following the syntax specified in section 2.2.32, with the following values:

- **CtlCode** MUST be set to FSCTL_SRV_REQUEST_RESUME_KEY.
- FileId.Persistent MUST be set to Open.DurableFileId. FileId.Volatile MUST be set to Open.FileId.
- InputOffset SHOULD be set to zero, and the receiver MUST ignore it. <128>
- InputCount MUST be set to 0.
- OutputOffset MUST be set to InputOffset + InputCount, rounded up to a multiple of 8.
- OutputCount MUST be set to 32.
- Flags MUST be set to 0.
- The server MUST copy the constructed SRV_REQUEST_RESUME_KEY that is used to identify the Open into the Buffer field at the OutputOffset computed above.

The response MUST be sent to the client.

3.3.5.15.6 Handling a Server-Side Data Copy Request

When the server receives a request with an SMB2 header with a Command value equal to SMB2 IOCTL, and a **CtlCode** of FSCTL_SRV_COPYCHUNK, message handling proceeds as follows:

The server MUST locate the source open from where data will be read by locating the Open using the SourceFile key received in the SRV_COPYCHUNK_COPY structure received in the buffer described by InputCount and InputOffset of the SMB2 IOCTL request. If the open is not found, the server MUST fail the request with STATUS_OBJECT_NAME_NOT_FOUND.

If Open.TreeConnect.Session of the destination file is not equal to Open.TreeConnect.Session of the source file, the server MUST fail the request with STATUS_OBJECT_NAME_NOT_FOUND.

If the server is configured to limit the amount of data copied in a single server side copy operation, and the size of data sent in the request exceeds the limit, the server MUST send an SMB2 IOCTL response as specified in the section Sending an Invalid Parameter Server Side Copy Response (section 3.3.5.15.6.2).

For each range, the server MUST issue a read using the SourceOffset and Length from the source file. If the SourceOffset or SourceOffset + Length extends beyond the end of file, the server SHOULD <129> treat this as a STATUS_END_OF_FILE error. If the read fails, the server MUST map the error code returned to a valid status code as described in section 2.2, and MUST construct a server copy failure response, as specified in the section Constructing a Server-Side Data Copy Response (section 3.3.5.15.6.2) and return it to the client.

The server MUST issue a write using the TargetOffset and Length in the range against the destination file. If the write fails, the server MUST construct a server copy failure response as specified in the section "Constructing a Server-Side Data Copy Response" and return it to the server.

If all ranges are copied successfully, the server MUST construct an SMB2 IOCTL response following the syntax specified in the section SMB2 IOCTL Response (section 2.2.32), with the following values:

- **CtlCode** MUST be set to FSCTL_SRV_COPYCHUNK.
- FileId.Persistent MUST be set to Open.DurableFileId. FileId.Volatile MUST be set to Open.FileId. This field MUST be the same as the FileId in the SMB2 IOCTL request.
- InputOffset SHOULD be set to zero, and the receiver MUST ignore it. <130>
- InputCount MUST be set to 0.
- OutputOffset MUST be set to InputOffset + InputCount, rounded up to a multiple of 8.
- OutputCount MUST be set to 12.
- Flags MUST be set to 0.
- The server MUST copy a SRV_COPYCHUNK_RESPONSE following the syntax specified in section SRV_COPYCHUNK_RESPONSE into the Buffer field at the OutputOffset computed above. ChunksWritten MUST be set to the number of chunks processed. ChunkBytesWritten MUST be set to 0. TotalBytesWritten MUST be set to the total number of bytes written to the destination file across all chunk writes.

The response MUST be sent to the client.

3.3.5.15.6.1 Constructing a Server-Side Data Copy Response

If a range is encountered that is not copied successfully, the server MUST construct an SMB2 IOCTL response following the syntax specified in section SMB2 IOCTL Response (section 2.2.32), with the following values:

- Status in the SMB2 header MUST be set to the error code received from the read or write.
- **CtlCode** MUST be set to FSCTL_SRV_COPYCHUNK.
- FileId.Persistent MUST be set to Open.DurableFileId. FileId.Volatile MUST be set to Open.FileId. This field MUST be the same as the FileId in the SMB2 IOCTL request.
- InputOffset SHOULD be set to zero, and the receiver MUST ignore it. <131>
- InputCount MUST be set to 0.
- OutputOffset MUST be set to InputOffset + InputCount, rounded up to a multiple of 8.

- OutputCount MUST be set to 12.
- Flags MUST be set to 0.
- The server MUST copy a SRV_COPYCHUNK_RESPONSE following the syntax specified in section SRV_COPYCHUNK_RESPONSE (section 2.2.32.1) into the Buffer field at the OutputOffset computed above. ChunksWritten MUST be set to the number of chunks successfully written. ChunkBytesWritten MUST be set to the number of bytes partially written, if the error was encountered partway through a write. TotalBytesWritten MUST be set to the total number of bytes written to the destination file across all chunk writes.

The response MUST be sent to the client.

3.3.5.15.6.2 Sending an Invalid Parameter Server Side Copy Response

If the server determines a request exceeds its configured maximum data size, the server MUST construct an SMB2 IOCTL Response, following the syntax specified in section 2.2.32, with the following values:

- Status in the SMB2 header MUST be set to STATUS_INVALID_PARAMETER.
- **CtlCode** MUST be set to FSCTL_SRV_COPYCHUNK.
- FileId MUST be set to Open.FileId.
- InputOffset SHOULD be set to zero, and the receiver MUST ignore it. <132>
- InputCount MUST be set to 0.
- OutputOffset MUST be set to InputOffset + InputCount, rounded up to a multiple of 8.
- OutputCount MUST be set to 12.
- Flags MUST be set to 0.
- The server MUST copy a SRV_COPYCHUNK_RESPONSE, following the syntax specified in section 2.2.32.1, into the **Buffer** field at the OutputOffset computed above, with the following differences. ChunksWritten MUST be set to the maximum number of chunks the server will accept in a single operation. ChunkBytesWritten MUST be set to the maximum number of bytes the server will accept in a single chunk. TotalBytesWritten MUST be set to the maximum total number of bytes the server will accept for a server side copy operation.

The response MUST be sent to the client.<133>

3.3.5.15.7 Handling a Pass-Through Operation Request

When the server receives a request with an SMB2 header with a Command value equal to SMB2 IOCTL, and a **CtlCode** not listed above, the server MAY pass it through to the local object store. <134> A list of FSCTL and IOCTL operations is listed in [MS-FSCC] section 2.3. If a server receives a **CtlCode** that is not one of the above mentioned values, it MUST fail the request with STATUS_INVALID_DEVICE_REQUEST.

The server MUST pass the following to the underlying object store: CtlCode, the input buffer described by InputOffset and InputCount, the output buffer described by OutputOffset and OutputCount, the MaxOutputResponse as the maximum output buffer size in bytes for the response, and MaxInputResponse as the maximum input buffer size in bytes for the response.

If the operation fails, the server MUST fail the request returning the error code received from the object store.

If the operation succeeds, the server MUST then construct an SMB2 IOCTL response following the syntax specified in section 2.2.32, with the following values:

- **CtlCode** MUST be set to the **CtlCode** of the request.
- FileId.Persistent MUST be set to Open.DurableFileId. FileId.Volatile MUST be set to Open.FileId. This field MUST be the same as the FileId in the SMB2 IOCTL request.
- InputOffset SHOULD be set to zero, and the receiver MUST ignore it. <135>
- InputCount MUST be set to the number of input bytes the object store wants to return to the client.
- OutputOffset MUST be set to InputOffset + InputCount, rounded up to a multiple of 8.
- The server MUST set the OutputCount to the actual number of bytes returned by the underlying object store in the output buffer.
- Flags MUST be set to 0.
- The server MUST copy the input and output response bytes into the ranges in Buffer described by InputOffset/InputCount and OutputOffset/OutputCount.

The response MUST be sent to the client.

3.3.5.16 Receiving an SMB2 CANCEL Request

When the server receives a request with an SMB2 header with a Command value equal to SMB2 CANCEL, message handling proceeds as follows:

An SMB2 CANCEL request is the only request received by the server that is not signed and does not contain a sequence number that must be checked. Thus, the server MUST NOT process the received packet as specified in sections 3.3.5.2.2 and 3.3.5.2.3.

The server MUST locate the request being canceled by searching the Connection.RequestList. If SMB2_FLAGS_ASYNC_COMMAND is set in the flags field of the SMB2 header of the cancel request, the server MUST search for a request in Connection.AsyncCommandList that matches both the MessageId and the AsyncId of the incoming cancel request. If SMB2_FLAGS_ASYNC_COMMAND is not set, then the server MUST search for a request that matches the MessageId of the incoming cancel request.

If a request is not found, the server MUST stop processing for this cancel request. No response is sent.

If a request is found, the server MUST attempt to cancel the request that was found, referred to here as the target request, by calling into the underlying object store. If the target request is successfully canceled, the target request MUST be failed by sending an ERROR response packet as specified in section 2.2.2, with the status field of the SMB2 header (specified in section 2.2.1) set to STATUS_CANCELLED. If the target request is not successfully canceled, processing MUST continue as is typical for the target request. No response is sent to the cancel request.

The cancel request indicates that the client wants to get a response for the target request, whether successful or not. The server MUST expedite the cancellation request by following the above steps.

3.3.5.17 Receiving an SMB2 ECHO Request

When the server receives a request with an SMB2 header with a Command value equal to SMB2 ECHO, message handling proceeds as follows:

The server MUST construct an SMB2 ECHO response following the syntax specified in section 2.2.29 and MUST send it to the client.

3.3.5.18 Receiving an SMB2 QUERY_DIRECTORY Request

When the server receives a request with an SMB2 header with a Command value equal to SMB2 QUERY_DIRECTORY, message handling proceeds as follows:

The server MUST locate the session that is taking the action, using the SessionId in the SMB2 header of the request to do a lookup on the SessionTable. If no session is found, or if Session.Connection is not equal to the connection on which the request was received, the server MUST fail the request with STATUS_USER_SESSION_DELETED.

Next, the server MUST locate the tree connect by performing a lookup in the **Session.TreeConnectTable**, using the **TreeId** in the SMB2 header of the request. If no tree connect is found, the server MUST fail the request with STATUS_NETWORK_NAME_DELETED.

Next, the server MUST locate the open for the directory to be queried by performing a lookup in the **Session.OpenTable**, using the **FileId.Volatile** of the request as the lookup key. If no open is found, or if **Open.DurableFileId** is not equal to **FileId.Persistent**, the server MUST fail the request with STATUS_FILE_CLOSED.

If the open is not an open to a directory, the request MUST be failed with STATUS_NOT_SUPPORTED.

If Open.GrantedAccess does not include FILE_LIST_DIRECTORY, the operation MUST be failed with STATUS_ACCESS_DENIED.

The information classes supported are specified in [MS-FSCC] section 2.4. The supported levels for the query are:

- FileDirectoryInformation
- FileFullDirectoryInformation
- FileBothDirectoryInformation
- FileIdFullDirectoryInformation
- FileIdBothDirectoryInformation
- FileNamesInformation

If any other information class is specified in the FileInformationClass field of the SMB2 QUERY_DIRECTORY request, the server MUST fail the operation with STATUS_INVALID_INFO_CLASS. If the information class requested is not supported by the server, the server MUST fail the request with STATUS_NOT_SUPPORTED.

If SMB2_REOPEN is set in the Flags field of the SMB2 QUERY_DIRECTORY request, the server MUST set Open.EnumerationLocation to 0 and Open.EnumerationSearchPattern to an empty string.

If SMB2_RESTART_SCANS is set in the Flags field of the SMB2 QUERY_DIRECTORY request, the server MUST set Open.EnumerationLocation to 0.

If Open.EnumerationLocation is 0 and Open.EnumerationSearchPattern is an empty string, then Open.EnumerationSearchPattern MUST be set to the search pattern specified in the SMB2 QUERY_DIRECTORY by FileNameOffset and FileNameLength. It is valid for FileNameLength to be 0.

If SMB2_INDEX_SPECIFIED is set in the Flags field of the SMB2 QUERY_DIRECTORY request and the underlying object store supports resuming enumerations by index number, the server MUST set Open.EnumerationLocation to the FileIndex received in the SMB2 QUERY_DIRECTORY request. An underlying store MAY choose to support resuming enumerations by index number.<136>

If SMB2_INDEX_SPECIFIED is set and FileNameLength is not zero, the server MUST set Open.EnumerationSearchPattern to the search pattern specified in the request by FileNameOffset and FileNameLength.

The server MUST now enumerate the files and directories that are contained within the directory specified by Open.LocalOpen, starting at the index Open.EnumerationLocation. Each entry MUST be formed as specified in [MS-FSCC] section 2.4. The server MUST fill in entries up to the OutputBufferLength received in the client request. The server MUST only include entries that match Open.EnumerationSearchPattern. For an explanation of wildcard evaluation for search patterns, see [CIFS] section 3.3. If SMB2_RETURN_SINGLE_ENTRY is set in the Flags field of the request, the server MUST return at most a single entry.

If TreeConnect.Share.DoAccessBasedDirectoryEnumeration is TRUE and the object store supports security, the server MUST also exclude entries for which the user represented by Session.SecurityContext does not have FILE_READ_DATA or FILE_LIST_DIRECTORY access. Upon filling the buffer, the server MUST set Open.EnumerationLocation to the location of the next enumeration entry that did not fit in the buffer. If all entries were placed in the buffer, the server MUST set Open.EnumerationLocation to an invalid value indicating that the enumeration is complete.

If an error is encountered, the server MUST fail the request with the error code received from the underlying object store by sending an error response as specified in section 2.2.2.

If there are no entries to return and this was the initial query (Open.EnumerationLocation was 0 before querying the object store), the server MUST fail the request with STATUS_NO_SUCH_FILE.

If there are no entries to return and this was not the initial query (Open.EnumerationLocation was not 0 before querying the object store), the server MUST fail the request with STATUS_NO_MORE_FILES.

Otherwise, the server MUST construct an SMB2_RESP_QUERY_DIRECTORY response following the syntax specified in section 2.2.34, with the following values:

- OutputBufferOffset MUST be set to the offset in bytes from the beginning of the SMB2 header where the enumeration data is being placed, the offset to Buffer[].
- OutputBufferLength MUST be set to the length, in bytes, of the result of the enumeration.
- The enumeration data MUST be copied into Buffer[].

The response MUST be sent to the client.

3.3.5.19 Receiving an SMB2 CHANGE_NOTIFY Request

When the server receives a request that has an SMB2 header with a Command value equal to SMB2 CHANGE_NOTIFY, message handling proceeds as follows:

The server MUST locate the session that is taking the action, using the **SessionId** in the SMB2 header of the request to do a lookup on the SessionTable. If no session is found, or if **Session.Connection** is not equal to the connection on which the request was received, the server MUST fail the request with STATUS_USER_SESSION_DELETED.

Next, the server MUST locate the tree connect by performing a lookup in the **Session.TreeConnectTable**, using the **TreeId** in the SMB2 header of the request. If no tree connect is found, the server MUST fail the request with STATUS_NETWORK_NAME_DELETED.

Next, the server MUST locate the open on which the client is requesting a change notification by performing a lookup in the **Session.OpenTable**, using the **FileId.Volatile** of the request as the lookup key. If no open is found, or if **Open.DurableFileId** is not equal to **FileId.Persistent**, the server MUST fail the request with STATUS_FILE_CLOSED.

If the open is not an open to a directory, the request MUST be failed with STATUS_INVALID_PARAMETER.

If Open.GrantedAccess does not include FILE_LIST_DIRECTORY, the operation MUST be failed with STATUS_ACCESS_DENIED.

Because change notify operations are not guaranteed to complete within a deterministic amount of time, the server SHOULD handle this operation asynchronously as specified in section 3.3.4.2.<137>

If the underlying object store does not support change notifications, the server MUST fail this request with STATUS_NOT_SUPPORTED.

The server MUST register a change notification on the underlying object store for the directory that is specified by Open.LocalOpen, using the completion filter supplied in the CompletionFilter field of the client request. If SMB2_WATCH_TREE is set in the Flags field of the client request, the server MUST request that the change notify monitor all subtrees of the directory that is specified by Open.LocalOpen. The server MUST indicate the maximum amount of notification data that it can accept by passing in the OutputBufferLength that is received from the client. An OutputBufferLength of 0 indicates that the client is interested in the occurrence of an event but the client is not interested in the notification data details.

Change notification processing in the object store MUST be handled as specified in section 3.3.1.4. It is also outlined in [CIFS] section 4.3.7.

If the object store returns an error, the server MUST fail the request with the error code received.

If the object store returns success, the server MUST construct an SMB2 CHANGE_NOTIFY response following the syntax that is specified in section 2.2.36 with the following values:

- OutputBufferOffset MUST be set to the offset, in bytes, from the beginning of the SMB2 header where the enumeration data is being placed, the offset to Buffer[].
- OutputBufferLength MUST be set to the length, in bytes, of the result of the enumeration. It is valid for length to be 0, indicating a change occurred but it could not be fit within the buffer.
- The change data MUST be copied into Buffer[].

The response MUST be sent to the client.

3.3.5.20 Receiving an SMB2 QUERY_INFO Request

When the server receives a request with an SMB2 header with a Command value equal to SMB2 QUERY_INFO, message handling proceeds as follows:

The server MUST locate the session that is taking the action, using the SessionId in the SMB2 header of the request to do a lookup on the SessionTable. If no session is found, or if **Session.Connection** is not equal to the connection on which the request was received, the server MUST fail the request with STATUS_USER_SESSION_DELETED.

Next, the server MUST locate the tree connect by performing a lookup in the **Session.TreeConnectTable**, using the **TreeId** in the SMB2 header of the request. If no tree connect is found, the server MUST fail the request with STATUS_NETWORK_NAME_DELETED.

Next, the server MUST locate the open on which the client is requesting the information by performing a lookup in the **Session.OpenTable**, using the **FileId.Volatile** of the request as the lookup key. If no open is found, or if **Open.DurableFileId** is not equal to **FileId.Persistent**, the server MUST fail the request with STATUS_FILE_CLOSED.

The remaining processing for this request depends on the InfoType that is requested and described below.

3.3.5.20.1 Handling SMB2_0_INFO_FILE

The information classes that are supported for querying files are listed in [MS-FSCC] section 2.4. Requests for information classes not listed in section 2.2.37 of [MS-SMB2] MUST be failed with STATUS_INVALID_INFO_CLASS.

If the object store supports security and the information class is FileBasicInformation, FileAllInformation, FilePipeInformation, FilePipeLocalInformation, FilePipeRemoteInformation, FileNetworkOpenInformation, or FileAttributeTagInformation, and Open.GrantedAccess does not include FILE_READ_ATTRIBUTES, the server MUST fail the request with STATUS_ACCESS_DENIED.

If the object store supports security and the information class is FileFullEaInformation and Open.GrantedAccess does not include FILE_READ_EA, the server MUST fail the request with STATUS_ACCESS_DENIED.

The server MUST query the information requested from the underlying object store. If the store does not support the data requested, the server MUST fail the request with STATUS_NOT_SUPPORTED.

Depending on the information class, the output data consists of a fixed portion followed by optional variable length data. If the OutputBufferLength given in the client request is insufficient to hold the fixed length part of the information requested, the server MUST fail the request with STATUS_INFO_LENGTH_MISMATCH and MUST return error data as specified in section 2.2.2 with ByteCount set to 0.

If the underlying object store returns an error, the server MUST fail the request with the error code received.

If the underlying object store returns only a portion of the variable-length data, the server MUST construct a response as described below but set the Status in the SMB2 header to STATUS_BUFFER_OVERFLOW. If FileFullEaInformation is being queried and the requested entries will not all fit in the Buffer field of the response, the server MUST construct a response as described below but set the Status in the SMB2 header to STATUS_BUFFER_OVERFLOW.

If the underlying object store returns the information successfully, the server MUST construct an SMB2 QUERY_INFO response with the following values:

- OutputBufferOffset MUST be set to the offset, in bytes, from the beginning of the SMB2 header to the attribute data at Buffer[].
- OutputBufferLength MUST be set to the length of the attribute data being returned to the client.
- The data MUST be placed in the response in Buffer[].

The response MUST then be sent to the client.

3.3.5.20.2 Handling SMB2_0_INFO_FILESYSTEM

The information classes that are supported for querying file systems are listed in [MS-FSCC] section 2.5.

Requests for other information classes not listed in section 2.2.37 of [MS-SMB2] MUST be failed with STATUS_INVALID_INFO_CLASS.

The server MUST query the information requested from the underlying volume that hosts the open in the object store. If the store does not support the data requested, the server MUST fail the request with STATUS_NOT_SUPPORTED.

Depending on the information class, the output data consists of a fixed portion followed by optional variable-length data. If the OutputBufferLength given in the client request is insufficient to hold the fixed length part of the information requested, the server MUST fail the request with STATUS_INFO_LENGTH_MISMATCH and MUST return error data, as specified in section 2.2.2 with ByteCount set to 0.

If the underlying object store returns an error, the server MUST fail the request with the error code received.

If the underlying object store returns only a portion of the variable-length data, the server MUST construct a success response as described below but set the Status in the SMB2 header to STATUS_BUFFER_OVERFLOW.

If the underlying object store returns the information successfully, the server MUST construct an SMB2 QUERY_INFO response with the following values:

- OutputBufferOffset MUST be set to the offset, in bytes, from the beginning of the SMB2 header to the attribute data at Buffer[].
- OutputBufferLength MUST be set to the length of the attribute data being returned to the client.
- The data MUST be placed in the response in Buffer[].

The response MUST then be sent to the client.

3.3.5.20.3 Handling SMB2_0_INFO_SECURITY

The following section assumes knowledge about Windows security concepts, as specified in [MS-SECO] and [MS-DTYP].

1. If the underlying object store does not support object security based on Access Control Lists (as specified in [MS-DTYP] section **2.4.5**), the server MUST return **WorldSid** for OWNER_SECURITY_INFORMATION and GROUP_SECURITY_INFORMATION and a NULL value for

DACL_SECURITY_INFORMATION, SACL_SECURITY_INFORMATION and LABEL_SECURITY_INFORMATION.

2. If SACL_SECURITY_INFORMATION is set in the SecurityInformation field of the request, and Open.GrantedAccess does not include ACCESS_SYSTEM_SECURITY, the server MUST fail the request with STATUS_ACCESS_DENIED.

3. If one of DACL_SECURITY_INFORMATION or LABEL_SECURITY_INFORMATION or GROUP_SECURITY_INFORMATION or OWNER_SECURITY_INFORMATION is set in the SecurityInformation field of the request, and Open.GrantedAccess does not include READ_CONTROL, the server MUST fail the request with STATUS_ACCESS_DENIED.

4. The server MUST call into the underlying object store to query the security descriptor for the object.

The fields required in the resulting security descriptor are denoted by the flags given in the SecurityInformation field of the request.

If the OutputBufferLength given in the client request is insufficient to hold the information requested, the server MUST fail the request with STATUS_INFO_LENGTH_MISMATCH and MUST return error data, as specified in section 2.2.2 with ByteCount set to 0. The server MUST NOT return STATUS_BUFFER_OVERFLOW with an incomplete security descriptor to the client as in the previous cases. If the underlying object store returns an error, the server MUST fail the request with the error code received.

If the underlying object store returns the information successfully, the server MUST construct an SMB2 QUERY_INFO response with the following values:

- OutputBufferOffset MUST be set to the offset, in bytes, from the beginning of the SMB2 header to the attribute data at Buffer[].
- OutputBufferLength MUST be set to the length of the attribute data being returned to the client.
- The security descriptor MUST be placed in the response in Buffer[].

The response MUST then be sent to the client.

3.3.5.20.4 Handling SMB2_0_INFO_QUOTA

The server's object store MAY support quotas that are associated with a security principal. If the server exposes support for quotas, it MUST allow security principals to be identified using security identifiers (SIDs) in the format that is specified in [MS-DTYP] section **2.4.2**.<138>

If the underlying object store does not support user quotas, the server MUST fail the request with STATUS_NOT_SUPPORTED.

The server MUST verify that the InputBufferOffset and InputBufferLength of the client request describe an SMB2_QUERY_QUOTA_INFO structure following the syntax specified in section 2.2.37.1. If not, the server MUST fail the request with STATUS_INVALID_PARAMETER.

The server MUST query the quota information that is requested from the underlying volume that hosts the open in the object store. If the **StartSidLength** is nonzero, then the server MUST fetch the quota information for the single SID that is specified by the **StartSidOffset**. If the **SidListLength** is nonzero and the **StartSidLength** is zero, then the server must enumerate the quota information for all the SIDs specified in the SidList specified by the **StartSidOffset**. The Sidlist MUST be a linked list of FILE_GET_QUOTA_INFORMATION structures. Otherwise the server MUST fail the request with STATUS_INVALID_PARAMETER.

If the OutputBufferLength given in the client request is insufficient to hold the information requested, the server MUST fail the request with STATUS_INFO_LENGTH_MISMATCH and MUST return error data, as specified in section 2.2.2 with ByteCount set to 0.

If the underlying object store returns an error, the server MUST fail the entire request with the error code received.

If the underlying object store returns only a portion of the data available, the server MUST construct a response, as described below, but set the Status in the SMB2 header to STATUS_BUFFER_OVERFLOW.

If the underlying object store returns the information successfully, the server MUST construct an SMB2 QUERY_INFO response with the following values:

- OutputBufferOffset MUST be set to the offset, in bytes, from the beginning of the SMB2 header to the attribute data at Buffer[].
- OutputBufferLength MUST be set to the length of the attribute data being returned to the client.
- The data MUST be placed in the response in Buffer[].

The response MUST then be sent to the client.

3.3.5.21 Receiving an SMB2 SET_INFO Request

When the server receives a request with an SMB2 header with a Command value equal to SMB2 SET_INFO, message handling proceeds as follows:

The server MUST locate the session taking the action using the SessionId in the SMB2 header of the request to do a lookup on the SessionTable. If no session is found, or if Session.Connection is not equal to the connection on which the request was received, the server MUST fail the request with STATUS_USER_SESSION_DELETED.

Next, the server MUST locate the tree connect by performing a lookup in Session.TreeConnectTable using the **TreeId** in the SMB2 header of the request. If no tree connect is found, the server MUST fail the request with STATUS_NETWORK_NAME_DELETED.

Next, the server MUST locate the open on which the client is requesting to set information by performing a lookup in **Session.OpenTable** using **FileId.Volatile** of the request as the lookup key. If no open is found, or if **Open.DurableFileId** is not equal to **FileId.Persistent**, the server MUST fail the request with STATUS_FILE_CLOSED.

The remaining processing for this request depends on the InfoType requested, as described below.

3.3.5.21.1 Handling SMB2_0_INFO_FILE

The information classes that are supported for applications using this operation are listed in [MS-FSCC] section 2.4. Requests for information classes not listed in section 2.2.39 MUST be failed with STATUS_INVALID_INFO_CLASS.

If the object store supports security and the information class is FileBasicInformation or FilePipeInformation, and Open.GrantedAccess does not include FILE_WRITE_ATTRIBUTES, the server MUST fail the request with STATUS_ACCESS_DENIED.

If the object store supports security and the information class is FileRenameInformation, FileDispositionInformation, or FileShortNameInformation, and Open.GrantedAccess does not include DELETE, the server MUST fail the request with STATUS_ACCESS_DENIED.

If the object store supports security and the information class is FileFullEaInformation, and Open.GrantedAccess does not include FILE_WRITE_EA, the server MUST fail the request with STATUS_ACCESS_DENIED.

If the object store supports security and the information class is FileAllocationInformation, FileEndOfFileInformation, or FileValidDataLengthInformation, and Open.GrantedAccess does not include FILE_WRITE_DATA, the server MUST fail the request with STATUS_ACCESS_DENIED.

The server MUST apply the information requested to the underlying object store. If the store does not support the information class requested, the server MUST fail the request with STATUS_NOT_SUPPORTED.

If the underlying object store returns an error, the server MUST fail the request with the error code received.

Otherwise, the server MUST initialize an SMB2 SET_INFO response following the syntax given in section 2.2.40.

The response MUST then be sent to the client.

3.3.5.21.2 Handling SMB2_0_INFO_FILESYSTEM

This InfoType is not valid for a set operation. The server MUST fail the operation with STATUS_INVALID_INFO_CLASS.

3.3.5.21.3 Handling SMB2_0_INFO_SECURITY

The following section assumes knowledge about Windows security concepts as specified in [MS-SECO] and [MS-DTYP]. <139>

1. If SACL_SECURITY_INFORMATION is set in the SecurityInformation field of the request, and Open.GrantedAccess does not include ACCESS_SYSTEM_SECURITY, the server MUST fail the request with STATUS_ACCESS_DENIED.

2. If DACL_SECURITY_INFORMATION is set in the SecurityInformation field of the request, and Open.GrantedAccess does not include WRITE_DAC, the server MUST fail the request with STATUS_ACCESS_DENIED.

3. If the object store supports security, either LABEL_SECURITY_INFORMATION, GROUP_SECURITY_INFORMATION, or OWNER_SECURITY_INFORMATION is set in the SecurityInformation field of the request, and Open.GrantedAccess does not include WRITE_OWNER, the server MUST fail the request with STATUS_ACCESS_DENIED.

4. The server MUST call into the underlying object store to set the security on the object.

The fields being applied in the provided security descriptor are denoted by the flags given in the SecurityInformation field of the request.

If the underlying object store returns an error, the server MUST fail the request with the error code received.

Otherwise, the server MUST initialize an SMB2 SET_INFO response following the syntax given in section 2.2.40.

The response MUST then be sent to the client.

3.3.5.21.4 Handling SMB2_0_INFO_QUOTA

The server's object store MAY support quotas associated with a security principal. If the server exposes support for quotas, it MUST allow security principals to be identified using security identifiers (SIDs) in the format specified in [MS-DTYP] section **2.4.2**.<140>

If the object store does not support quotas, the server MUST fail the request with STATUS_NOT_SUPPORTED.

If the user represented by Session.SecurityContext is not granted the right to manage quotas on the underlying volume in the object store, the server MUST fail the request with STATUS_ACCESS_DENIED.

The server MUST apply the provided quota information to the underlying volume that hosts the open in the object store.

If the underlying object store returns an error, the server MUST fail the request with the error code received.

Otherwise, the server MUST initialize an SMB2 SET_INFO response following the syntax given in section 2.2.40.

The response MUST then be sent to the client.

3.3.5.22 Receiving an SMB2 OPLOCK_BREAK Acknowledgment

When the server receives a request with an SMB2 header with a Command value equal to SMB2 OPLOCK_BREAK, message handling proceeds as follows:

The server MUST locate the session taking the action using the SessionId in the SMB2 header of the request to do a lookup on the SessionTable. If no session is found, or if Session.Connection is not equal to the connection on which the request was received, the server MUST fail the request with STATUS_USER_SESSION_DELETED.

Next, the server MUST locate the tree connect by performing a lookup in Session.TreeConnectTable using the **TreeId** in the SMB2 header of the request. If no tree connect is found, the server MUST fail the request with STATUS_NETWORK_NAME_DELETED.

Next, the server MUST locate the open on which the client is acknowledging an oplock break by performing a lookup in **Session.OpenTable** using **FileId.Volatile** of the request as the lookup key. If no open is found, or if **Open.DurableFileId** is not equal to **FileId.Persistent**, the server MUST fail the request with STATUS_FILE_CLOSED.

If Open.OplockState is not Breaking, the server MUST fail the request with STATUS_INVALID_DEVICE_STATE.

If the OplockLevel of the request is not SMB2_OPLOCK_LEVEL_NONE or SMB2_OPLOCK_LEVEL_II, the server MUST fail the request with STATUS_INVALID_OPLOCK_PROTOCOL.

If Open.OplockLevel is not Exclusive or Batch, and the OplockLevel of the request is SMB2_OPLOCK_LEVEL_II, the server MUST fail the request with STATUS_INVALID_OPLOCK_PROTOCOL.

The server completes the oplock break request received from the object store as described in section 3.3.4.5. If the OplockLevel of the request is SMB2_OPLOCK_LEVEL_II, the server MUST set Open.OplockLevel to SMB2_OPLOCK_LEVEL_II and Open.OplockState to Held. If the OplockLevel of

the request is SMB2_OPLOCK_LEVEL_NONE, the server MUST set Open.OplockLevel to SMB2_OPLOCK_LEVEL_NONE and Open.OplockState to None.

The server then MUST construct an oplock break response using the syntax specified in section 2.2.25 with the following values:

- OplockLevel MUST be set to Open.OplockLevel.

This response MUST then be sent to the client.

3.3.6 Timer Events

3.3.6.1 Oplock Break Acknowledgment Timer Event

The oplock break acknowledgment timer MUST be started when the server sends an SMB2 OPLOCK_BREAK notification (as specified in section 2.2.23) to the client as a result of the underlying object store indicating an oplock break on a file. When the oplock break acknowledgment timer expires, the server MUST scan for oplock breaks that have not been acknowledged by the client within the configured time. It does this by enumerating all opens in the GlobalOpenTable. For each open, if Open.OplockState is Breaking and Open.OplockTimeout is earlier than the current time, the server MUST acknowledge the oplock break to the underlying object store represented by Open.LocalOpen, set Open.OplockLevel to SMB2_OPLOCK_LEVEL_NONE, and set Open.OplockState to None.

The timer MUST then be restarted to expire again at the time of the next oplock time-out. If no other opens have Open.OplockState equal to Breaking, the timer MUST NOT be restarted.

3.3.6.2 Durable Open Scavenger Timer Event

The durable open scavenger timer MUST be started when the connection or session associated with a file is disconnected. When the durable open scavenger timer expires, the server MUST scan for durable opens that have not been reclaimed by a client within the configured time. It does this by enumerating all opens in the GlobalOpenTable. For each open, if Open.Connection is NULL and Open.DurableOpenTimeout is earlier than the current time, the server MUST close Open.LocalOpen, remove the open from the GlobalOpenTable, and free the open object.

The timer MUST then be restarted to expire again at the time of the next durable open time-out. If no other opens have Open.Connection equal to NULL, the timer MUST NOT be restarted.

3.3.6.3 Session Expiration Timer Event

When the session expiration timer expires, the server MUST walk each Session in the SessionTable for each Connection. If the Session.State is Active and the Session.ExpirationTime has passed, the Session.State MUST be set to Expired.

3.3.7 Other Local Events

3.3.7.1 Handling a Transport Disconnect

When the transport indicates a disconnect, the server MUST enumerate the sessions established over this connection in Connection.SessionTable.

For every session in the table, the server MUST free every tree connect object in Session.TreeConnectTable. For every open in Session.OpenTable, if Open.OplockLevel is equal to Batch, Open.OplockState is equal to Held, and Open.IsDurable is TRUE, the open MUST be

preserved for a reconnect as described in section 3.3.5.9.7. The open MUST be removed from Session.OpenTable, Open.Connection is set to NULL, and Open.Session is set to NULL.

If Open.OplockLevel is not equal to Batch, or Open.OplockState is not equal to Held, or Open.IsDurable is not TRUE, then Open.LocalOpen MUST be closed. The open is removed from Session.OpenTable, and the open is freed. Then the session MUST be removed from Connection.SessionTable and freed.

Any requests in Connection.RequestList MUST be canceled and freed.

Finally, the connection MUST be freed.

4 Protocol Examples

The following sections describe common scenarios that indicate normal traffic flow in order to illustrate the function of the Server Message Block (SMB) Version 2.0 Protocol.

4.1 Connecting to a Share by Using a Multi-Protocol Negotiate

The following diagram shows the steps taken by a client that is negotiating SMB2 by using an SMB-style negotiate.

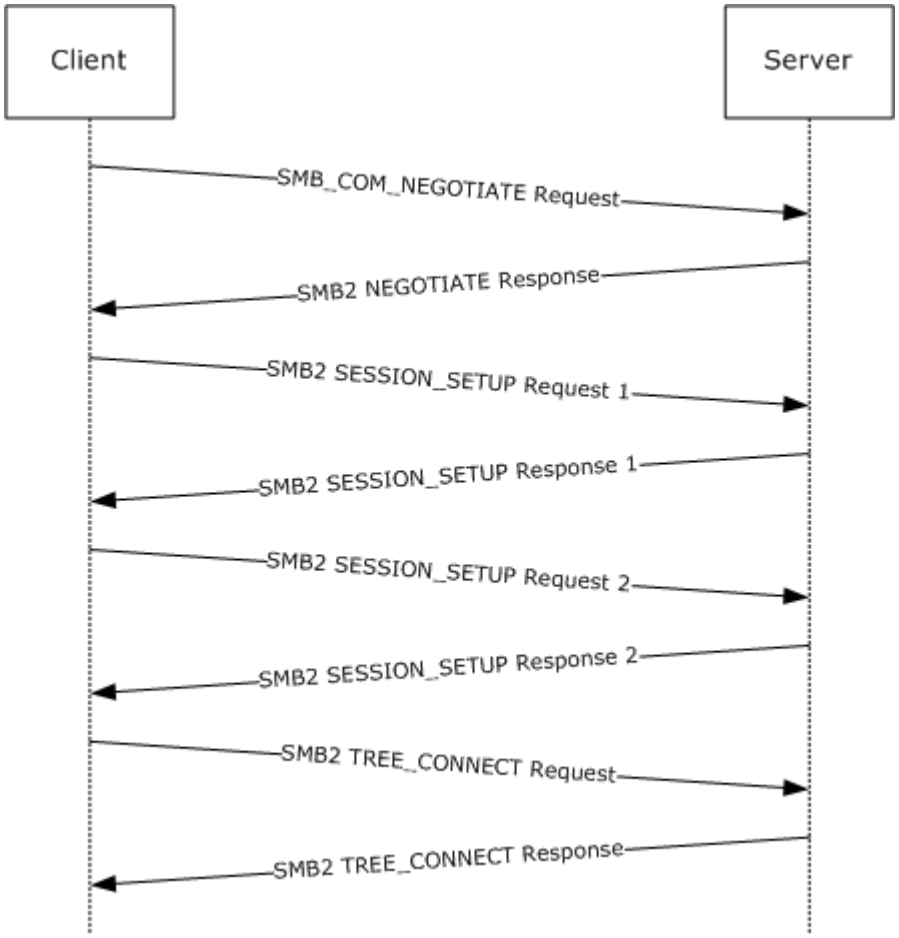


Figure 3: Client negotiating SMB2 with SMB-style negotiate

1. The client sends an SMB negotiate packet with the string "SMB 2.002" in the dialect string list, along with the other SMB dialects the client supports.

```
Smb: C; Negotiate, Dialect = PC NETWORK PROGRAM 1.0, LANMAN1.0, Windows for Workgroups
3.1a, LM1.2X002, LANMAN2.1, NT LM 0.12, SMB 2.002
Protocol: SMB
Command: Negotiate 114(0x72)
SMBHeader: Command, TID: 0xFFFF, PID: 0xFEFF, UID: 0x0000, MID: 0x0000
Flags: 24 (0x18)
Bit0: (.......0) SMB FLAGS LOCK AND READ OK: LOCK AND READ and WRITE AND CLOSE not
supported (obsoleted)
```

```
Bit1: (......0.) SMB FLAGS SEND NO ACK [not implemented]
Bit2: (.....0..) Reserved (must be zero)
Bit3: (....1...) SMB_FLAGS_CASE_INSENSITIVE: SMB paths are case-insensitive
Bit4: (...1....) SMB FLAGS CANONICALIZED PATHS: Canonicalized File and pathnames
(obsoleted)
Bit5: (..0.....) SMB FLAGS OPLOCK: No Oplocks supported for OPEN, CREATE & CREATE NEW
(obsoleted)
Bit6: (.0......) SMB_FLAGS_OPLOCK_NOTIFY_ANY: No Notifications supported for OPEN,
CREATE & CREATE_NEW (obsoleted)
Bit7: (0.......) SMB FLAGS SERVER TO REDIR: Command - SMB is being sent from the
client
Flags2: 51283 (0xC853)
Bit00: (...............1) SMB FLAGS2 KNOWS LONG NAMES: May return long file names
Bit01: (..............1.) SMB_FLAGS2_KNOWS_EAS: Understands extended attributes
Bit02: (.............0..) SMB_FLAGS2_SMB_SECURITY_SIGNATURE: Not security signature-
enabled
Bit03: (............0...) Reserved
Bit04: (...........1....) Reserved
Bit05: (..........0.....) SMB FLAGS2 SMB SECURITY SIGNATURE REQUIRED: SMB packets must
be signed
Bit06: (.........1......) SMB FLAGS2 IS LONG NAME: Any path name in the request is a
long name
Bit07: (........0.......) Reserved
Bit08: (.......0........) Reserved
Bit09: (......0.........) Reserved
Bit10: (.....0..........) SMB_FLAGS2_REPARSE_PATH: Not requesting Reparse path
Bit11: (....1...........) SMB_FLAGS2_EXTENDED_SECURITY: Aware of extended security
Bit12: (...0............) SMB FLAGS2 DFS: No DFS namespace
Bit13: (..0.............) SMB FLAGS2 PAGING IO: Read operation will NOT be permitted
if has no read permission
Bit14: (.1..............) SMB FLAGS2 NT STATUS: Using 32-bit NT status error codes
Bit15: (1...............) SMB_FLAGS2_UNICODE: Using UNICODE strings
PIDHigh: 0 (0x0)
SecuritySignature: 0x0
Reserved: 0 (0x0)
TreeID: 65535 (0xFFFF)
ProcessID: 65279 (0xFEFF)
UserID: 0 (0x0)
MultiplexID: 0 (0x0)
CNegotiate:
WordCount: 0 (0x0)
ByteCount: 109 (0x6D)
Dialect: PC NETWORK PROGRAM 1.0
BufferFormat: Dialect 2(0x2)
DialectName: PC NETWORK PROGRAM 1.0
Dialect: LANMAN1.0
BufferFormat: Dialect 2(0x2)
DialectName: LANMAN1.0
Dialect: Windows for Workgroups 3.1a
BufferFormat: Dialect 2(0x2)
DialectName: Windows for Workgroups 3.1a
Dialect: LM1.2X002
BufferFormat: Dialect 2(0x2)
DialectName: LM1.2X002
Dialect: LANMAN2.1
BufferFormat: Dialect 2(0x2)
DialectName: LANMAN2.1
Dialect: NT LM 0.12
BufferFormat: Dialect 2(0x2)
DialectName: NT LM 0.12
Dialect: SMB 2.002
BufferFormat: Dialect 2(0x2)
DialectName: SMB 2.002
```

2. The server receives the SMB negotiate request and finds dialect "SMB 2.002". The server responds with an SMB2 negotiate.

```
Smb2: R NEGOTIATE
SMB2Header:
Size: 64 (0x40)
Epoch: 0 (0x0)
Status: STATUS_SUCCESS
Command: NEGOTIATE
Credits: 1 (0x1)
Flags: 1 (0x1)
ServerToRedir: ...............................1  Server to Client
AsyncCommand:  ...............................0.  Command is not asynchronous
Related:       ...............................0..  Packet is single message
Signed:        ...............................0...  Packet is not signed
Reserved: 0 (0x0)
DFS:           0.............................  Command is not a DFS Operation
NextCommand: 0 (0x0)
MessageId: 0 (0x0)
ProcessId: 65279 (0xFEFF)
TreeId: 0 (0x0)
SessionId: 0 (0x0)
RNegotiate:
Size: 65 (0x41)
SecurityMode: Signing Enabled
DialectRevision: 0x0202
Reserved: 0 (0x0)
Guid: {3F5CF209-A4E5-0049-A7D6-6A456D5CA5CF}
Capabilities: 1 (0x1)
DFS:           ...............................1  DFS available
MaxTransactSize: 65536 (0x10000)
MaxReadSize: 65536 (0x10000)
MaxWriteSize: 65536 (0x10000)
SystemTime: 127972992061679232 (0x1C6A6C21CAE2680)
ServerStartTime: 127972985895467232 (0x1C6A6C0AD2538E0)
SecurityBufferOffset: 128 (0x80)
SecurityBufferLength: 30 (0x1E)
Reserved2: 0 (0x0)
Buffer:
```

3. The client queries GSS for the authentication token and sends an SMB2 session setup request with the output token received from GSS.

```
Smb2: C SESSION SETUP
Smb2: C SESSION SETUP
SMB2Header:
Size: 64 (0x40)
Epoch: 0 (0x0)
Status: STATUS_SUCCESS
Command: SESSION SETUP
Credits: 126 (0x7E)
Flags: 0 (0x0)
ServerToRedir: ...............................0  Client to Server
AsyncCommand:  ...............................0.  Command is not asynchronous
Related:       ...............................0..  Packet is single message
Signed:        ...............................0...  Packet is not signed
```

```
Reserved: 0 (0x0)
DFS:           0.............................  Command is not a DFS Operation
NextCommand: 0 (0x0)
MessageId: 1 (0x1)
ProcessId: 65279 (0xFEFF)
TreeId: 0 (0x0)
SessionId: 0 (0x0)
CSessionSetup:
Size: 25 (0x19)
VcNumber: 0 (0x0)
SecurityMode: Signing Enabled
Capabilities: 1 (0x1)
DFS:            .............................1 DFS available
Channel: 0 (0x0)
SecurityBufferOffset: 88 (0x58)
SecurityBufferLength: 74 (0x4A)
Buffer: (74 bytes)
```

4. The server processes the token received with GSS and gets a return code indicating a subsequent round trip is required. The server responds to the client with an SMB2 session setup response with Status equal to STATUS_MORE_PROCESSING_REQUIRED and the response containing the output token from GSS.

```
Smb2: R SESSION SETUP (Status=STATUS MORE PROCESSING REQUIRED)
Smb2: R SESSION SETUP (Status=STATUS_MORE_PROCESSING_REQUIRED)
SMB2Header:
Size: 64 (0x40)
Epoch: 0 (0x0)
Status: STATUS MORE PROCESSING REQUIRED
Command: SESSION SETUP
Credits: 2 (0x2)
Flags: 1 (0x1)
ServerToRedir: ..............................1  Server to Client
AsyncCommand:  ..............................0.  Command is not asynchronous
Related:       ..............................0..  Packet is single message
Signed:        ..............................0...  Packet is not signed
Reserved: 0 (0x0)
DFS:           0.............................  Command is not a DFS Operation
NextCommand: 0 (0x0)
MessageId: 1 (0x1)
ProcessId: 65279 (0xFEFF)
TreeId: 0 (0x0)
SessionId: 4398046511113 (0x40000000009)
RSessionSetup:
Size: 9 (0x9)
SessionFlags: Normal session
SecurityBufferOffset: 72 (0x48)
SecurityBufferLength: 219 (0xDB)
Buffer: (219 bytes)
```

5. The client processes the received token with GSS and sends an SMB2 session setup request with the output token received from GSS and the SessionId received on the previous response.

```
Smb2: C SESSION SETUP
Smb2: C SESSION SETUP
SMB2Header:
```

```
Size: 64 (0x40)
Epoch: 0 (0x0)
Status: STATUS_SUCCESS
Command: SESSION SETUP
Credits: 125 (0x7D)
Flags: 0 (0x0)
ServerToRedir: ..............................0  Client to Server
AsyncCommand:  ..............................0.  Command is not asynchronous
Related:       ..............................0..  Packet is single message
Signed:        ..............................0...  Packet is not signed
Reserved: 0 (0x0)
DFS:           0.............................  Command is not a DFS Operation
NextCommand: 0 (0x0)
MessageId: 2 (0x2)
ProcessId: 65279 (0xFEFF)
TreeId: 0 (0x0)
SessionId: 4398046511113 (0x400000000009)
CSessionSetup:
Size: 25 (0x19)
VcNumber: 0 (0x0)
SecurityMode: Signing Enabled
Capabilities: 1 (0x1)
DFS:           ..............................1 DFS available
Channel: 0 (0x0)
SecurityBufferOffset: 88 (0x58)
SecurityBufferLength: 245 (0xF5)
Buffer: (245 bytes)
```

6. The server processes the token received with GSS and gets a successful return code. The server responds to client with an SMB2 session setup response with Status equal to STATUS_SUCCESS and the response containing the output token from GSS.

```
Smb2: R SESSION SETUP
Smb2: R SESSION SETUP
SMB2Header:
Size: 64 (0x40)
Epoch: 0 (0x0)
Status: STATUS_SUCCESS
Command: SESSION SETUP
Credits: 3 (0x3)
Flags: 9 (0x9)
ServerToRedir: ..............................1  Server to Client
AsyncCommand:  ..............................0.  Command is not asynchronous
Related:       ..............................0..  Packet is single message
Signed:        ..............................1...  Packet is signed
Reserved: 0 (0x0)
DFS:           0.............................  Command is not a DFS Operation
NextCommand: 0 (0x0)
MessageId: 2 (0x2)
ProcessId: 65279 (0xFEFF)
TreeId: 0 (0x0)
SessionId: 4398046511113 (0x400000000009)
RSessionSetup:
Size: 9 (0x9)
SessionFlags: Normal session
SecurityBufferOffset: 72 (0x48)
SecurityBufferLength: 29 (0x1D)
Buffer: (29 bytes)
```

7. The client completes the authentication and sends an SMB2 tree connect request with the SessionId for the session, and a tree connect request containing the Unicode share name "\\smb2server\IPC$".

```
Smb2: C TREE CONNECT \\smb2server\IPC$
SMB2Header:
Size: 64 (0x40)
Epoch: 0 (0x0)
Status: STATUS_SUCCESS
Command: TREE CONNECT
Credits: 123 (0x7B)
Flags: 0 (0x0)
ServerToRedir: ..............................0  Client to Server
AsyncCommand:  ..............................0.  Command is not asynchronous
Related:       ..............................0..  Packet is single message
Signed:        ..............................0...  Packet is not signed
Reserved: 0 (0x0)
DFS: 0.............................  Command is not a DFS Operation
NextCommand: 0 (0x0)
MessageId: 3 (0x3)
ProcessId: 65279 (0xFEFF)
TreeId: 0 (0x0)
SessionId: 4398046511113 (0x40000000009)
CTreeConnect:
Size: 9 (0x9)
Reserved: 0 (0x0)
PathOffset: 72 (0x48)
PathLength: 34 (0x22)
Share: \\smb2server\IPC$
```

8. The server responds with an SMB2 tree connect response with MessageId of 3, CreditsGranted of 5, Status equal to STATUS_SUCCESS, SessionId of 0x40000000001, and **TreeId** set to the locally generated identifier 0x1.

```
Smb2: R TREE CONNECT TID=0x1
SMB2Header:
Size: 64 (0x40)
Epoch: 0 (0x0)
Status: STATUS_SUCCESS
Command: TREE CONNECT
Credits: 5 (0x5)
Flags: 1 (0x1)
ServerToRedir: ..............................1  Server to Client
AsyncCommand:  ..............................0.  Command is not asynchronous
Related:       ..............................0..  Packet is single message
Signed:        ..............................0...  Packet is not signed
Reserved: 0 (0x0)
DFS:           0.............................  Command is not a DFS Operation
NextCommand: 0 (0x0)
MessageId: 3 (0x3)
ProcessId: 65279 (0xFEFF)
TreeId: 1 (0x1)
SessionId: 4398046511113 (0x40000000009)
RTreeConnect:
Size: 16 (0x10)
ShareType: Pipe
Reserved: 0 (0x0)
Flags: No Caching
Capabilities: 0 (0x0)
MaximalAccess: 2032127 (0x1F01FF)
```

Further operations can now continue, using the SessionId and **TreeId** generated in the connection to this share.

4.2 Connecting to a Share by Using an SMB2 Negotiate

The following diagram shows the steps taken by a client that is negotiating SMB2 by using an SMB2 negotiate.

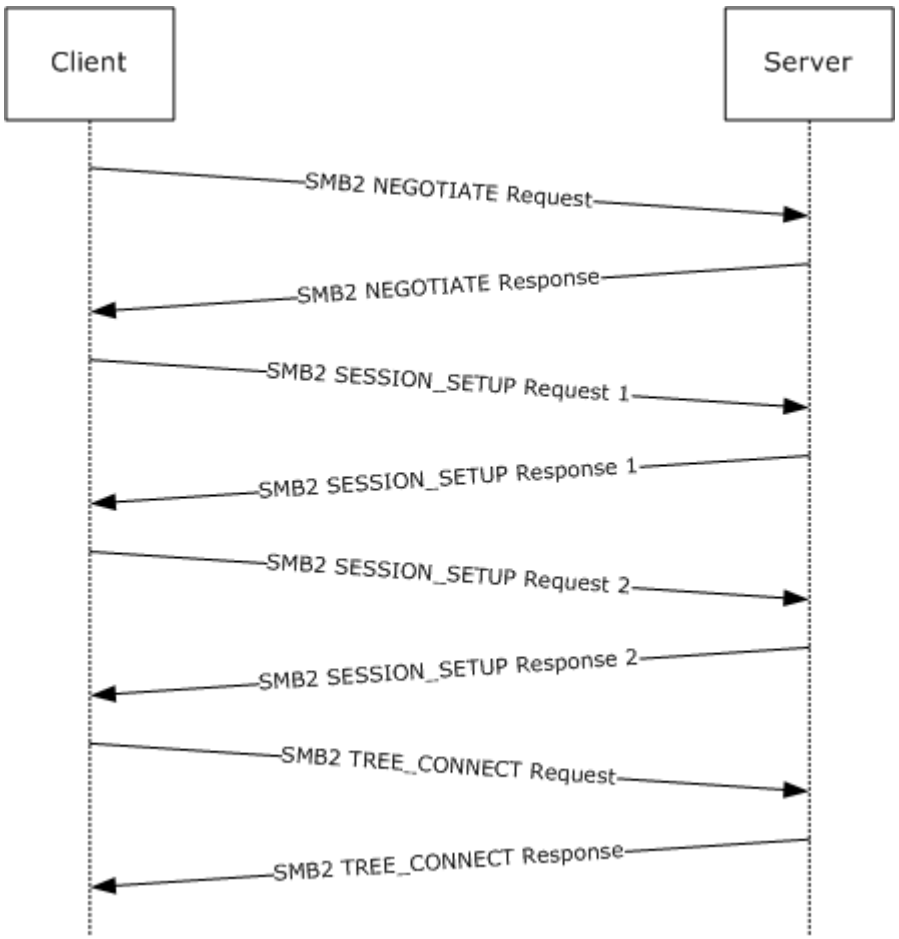


Figure 4: Client negotiating SMB2 with SMB2 negotiate

1. The client sends an SMB2 negotiate packet with the dialect "0" in the dialect array.

```
Smb2: C NEGOTIATE
SMB2Header:
Size: 64 (0x40)
Epoch: 0 (0x0)
Status: STATUS SUCCESS
Command: NEGOTIATE
Credits: 126 (0x7E)
```

```
Flags: 0 (0x0)
ServerToRedir: ..............................0  Client to Server
AsyncCommand:  ..............................0.  Command is not asynchronous
Related:       ..............................0..  Packet is single message
Signed:        ..............................0...  Packet is not signed
Reserved: 0 (0x0)
DFS:           0.............................  Command is not a DFS Operation
NextCommand: 0 (0x0)
MessageId: 0 (0x0)
ProcessId: 65279 (0xFEFF)
TreeId: 0 (0x0)
SessionId: 0 (0x0)

CNegotiate:
Size: 36 (0x24)
DialectCount: 1 (0x1)
SecurityMode: Signing Enabled
Reserved: 0 (0x0)
Capabilities: 0 (0x0)
Guid: {00000000-0000-0000-0000-000000000000}
StartTime: 0 (0x0)
Dialects: 0 (0x0)
```

2. The server receives the SMB2 negotiate request and finds dialect 0. The server responds with an SMB2 negotiate.

```
Smb2: R NEGOTIATE
SMB2Header:
Size: 64 (0x40)
Epoch: 0 (0x0)
Status: STATUS SUCCESS
Command: NEGOTIATE
Credits: 1 (0x1)
Flags: 1 (0x1)
ServerToRedir: ..............................1  Server to Client
AsyncCommand:  ..............................0.  Command is not asynchronous
Related:       ..............................0..  Packet is single message
Signed:        ..............................0...  Packet is not signed
Reserved: 0 (0x0)
DFS:           0.............................  Command is not a DFS Operation
NextCommand: 0 (0x0)
MessageId: 0 (0x0)
ProcessId: 65279 (0xFEFF)
TreeId: 0 (0x0)
SessionId: 0 (0x0)
RNegotiate:
Size: 65 (0x41)
SecurityMode: Signing Enabled
DialectRevision: 514 (0x0202)
Reserved: 0 (0x0)
Guid: {3F5CF209-A4E5-0049-A7D6-6A456D5CA5CF}
Capabilities: 1 (0x1)
DFS:           ..............................1  DFS available
MaxTransactSize: 65536 (0x10000)
MaxReadSize: 65536 (0x10000)
MaxWriteSize: 65536 (0x10000)
SystemTime: 127972992061679232 (0x1C6A6C21CAE2680)
ServerStartTime: 127972985895467232 (0x1C6A6C0AD2538E0)
SecurityBufferOffset: 128 (0x80)
SecurityBufferLength: 30 (0x1E)
Reserved2: 0 (0x0)
```

```
Buffer:
```

3. The client queries GSS for the authentication token and sends an SMB2 session setup request with the output token received from GSS.

```
Smb2: C SESSION SETUP
SMB2Header:
Size: 64 (0x40)
Epoch: 0 (0x0)
Status: STATUS SUCCESS
Command: SESSION SETUP
Credits: 126 (0x7E)
Flags: 0 (0x0)
ServerToRedir: ..............................0  Client to Server
AsyncCommand:  ..............................0.  Command is not asynchronous
Related:       ..............................0..  Packet is single message
Signed:        ..............................0...  Packet is not signed
Reserved: 0 (0x0)
DFS:           0.............................  Command is not a DFS Operation
NextCommand: 0 (0x0)
MessageId: 1 (0x1)
ProcessId: 65279 (0xFEFF)
TreeId: 0 (0x0)
SessionId: 0 (0x0)
CSessionSetup:
Size: 25 (0x19)
VcNumber: 0 (0x0)
SecurityMode: Signing Enabled
Capabilities: 1 (0x1)
DFS:           ..............................1 DFS available
Channel: 0 (0x0)
SecurityBufferOffset: 88 (0x58)
SecurityBufferLength: 74 (0x4A)
Buffer: (74 bytes)
```

4. The server processes the token received with GSS and gets a return code indicating a subsequent round trip is required. The server responds to the client with an SMB2 session setup response with Status equal to STATUS_MORE_PROCESSING_REQUIRED and the response containing the output token from GSS.

```
Smb2: R SESSION SETUP (Status=STATUS MORE PROCESSING REQUIRED)
SMB2Header:
Size: 64 (0x40)
Epoch: 0 (0x0)
Status: STATUS MORE PROCESSING REQUIRED
Command: SESSION SETUP
Credits: 2 (0x2)
Flags: 1 (0x1)
ServerToRedir: ..............................1  Server to Client
AsyncCommand:  ..............................0.  Command is not asynchronous
Related:       ..............................0..  Packet is single message
Signed:        ..............................0...  Packet is not signed
Reserved: 0 (0x0)
DFS:           0.............................  Command is not a DFS Operation
NextCommand: 0 (0x0)
MessageId: 1 (0x1)
```

```
ProcessId: 65279 (0xFEFF)
TreeId: 0 (0x0)
SessionId: 4398046511113 (0x40000000009)
RSessionSetup:
Size: 9 (0x9)
SessionFlags: Normal session
SecurityBufferOffset: 72 (0x48)
SecurityBufferLength: 219 (0xDB)
Buffer: (219 bytes)
```

5. The client processes the received token with GSS and sends an SMB2 session setup request with the output token received from GSS and the SessionId received on the previous response.

```
Smb2: C SESSION SETUP
SMB2Header:
Size: 64 (0x40)
Epoch: 0 (0x0)
Status: STATUS SUCCESS
Command: SESSION SETUP
Credits: 125 (0x7D)
Flags: 0 (0x0)
ServerToRedir: ..............................0  Client to Server
AsyncCommand:  ..............................0.  Command is not asynchronous
Related:       ..............................0..  Packet is single message
Signed:        ..............................0...  Packet is not signed
Reserved: 0 (0x0)
DFS:           0.............................  Command is not a DFS Operation
NextCommand: 0 (0x0)
MessageId: 2 (0x2)
ProcessId: 65279 (0xFEFF)
TreeId: 0 (0x0)
SessionId: 4398046511113 (0x40000000009)
CSessionSetup:
Size: 25 (0x19)
VcNumber: 0 (0x0)
SecurityMode: Signing Enabled
Capabilities: 1 (0x1)
DFS:           ..............................1 DFS available
Channel: 0 (0x0)
SecurityBufferOffset: 88 (0x58)
SecurityBufferLength: 245 (0xF5)
Buffer: (245 bytes)
```

6. The server processes the token received with GSS and gets a successful return code. The server responds to the client with an SMB2 session setup response with Status equal to STATUS_SUCCESS and the response containing the output token from GSS.

```
Smb2: R SESSION SETUP
SMB2Header:
Size: 64 (0x40)
Epoch: 0 (0x0)
Status: STATUS SUCCESS
Command: SESSION SETUP
Credits: 3 (0x3)
Flags: 9 (0x9)
ServerToRedir: ..............................1 Server to Client
```

```
AsyncCommand:  ..............................0.  Command is not asynchronous
Related:       ..............................0..  Packet is single message
Signed:        ............................1...  Packet is signed
Reserved: 0 (0x0)
DFS:           0.............................  Command is not a DFS Operation
NextCommand: 0 (0x0)
MessageId: 2 (0x2)
ProcessId: 65279 (0xFEFF)
TreeId: 0 (0x0)
SessionId: 4398046511113 (0x40000000009)
RSessionSetup:
Size: 9 (0x9)
SessionFlags: Normal session
SecurityBufferOffset: 72 (0x48)
SecurityBufferLength: 29 (0x1D)
Buffer: (29 bytes)
```

7. The client completes the authentication and sends an SMB2 tree connect request with the SessionId for the session, and a tree connect request containing the Unicode share name "\\smb2server\IPC$".

```
Smb2: C TREE CONNECT \\smb2server\IPC$
SMB2Header:
Size: 64 (0x40)
Epoch: 0 (0x0)
Status: STATUS_SUCCESS
Command: TREE CONNECT
Credits: 123 (0x7B)
Flags: 0 (0x0)
ServerToRedir: ..............................0  Client to Server
AsyncCommand:  ..............................0.  Command is not asynchronous
Related:       ..............................0..  Packet is single message
Signed:        ............................0...  Packet is not signed
Reserved: 0 (0x0)
DFS:           0.............................  Command is not a DFS Operation
NextCommand: 0 (0x0)
MessageId: 3 (0x3)
ProcessId: 65279 (0xFEFF)
TreeId: 0 (0x0)
SessionId: 4398046511113 (0x40000000009)
CTreeConnect:
Size: 9 (0x9)
Reserved: 0 (0x0)
PathOffset: 72 (0x48)
PathLength: 34 (0x22)
Share: \\smb2server\IPC$
```

8. The server responds with an SMB2 tree connect response with MessageId of 3, CreditsGranted of 5, Status equal to STATUS_SUCCESS, SessionId of 0x40000000001, and **TreeId** set to the locally generated identifier 0x1.

```
Smb2: R TREE CONNECT TID=0x1
SMB2Header:
Size: 64 (0x40)
Epoch: 0 (0x0)
```

```
Status: STATUS SUCCESS
Command: TREE CONNECT
Credits: 5 (0x5)
Flags: 1 (0x1)
ServerToRedir: ...............................1  Server to Client
AsyncCommand:  ...............................0.  Command is not asynchronous
Related:       ...............................0..  Packet is single message
Signed:        ...............................0...  Packet is not signed
Reserved: 0 (0x0)
DFS:           0.............................  Command is not a DFS Operation
NextCommand: 0 (0x0)
MessageId: 3 (0x3)
ProcessId: 65279 (0xFEFF)
TreeId: 1 (0x1)
SessionId: 4398046511113 (0x40000000009)
RTreeConnect:
Size: 16 (0x10)
ShareType: Pipe
Reserved: 0 (0x0)
Flags: No Caching
Capabilities: 0 (0x0)
MaximalAccess: 2032127 (0x1F01FF)
```

Further operations can now continue, using the SessionId and **TreeId** generated in the connection to this share.

4.3 Executing an Operation on a Named Pipe

The following diagram demonstrates the steps taken to execute transactions over a named pipe using both individual reads and writes, and the transact named pipe operation. Assume that this sequence starts on a connection where the session and tree connect have been established as described in previous sections with SessionId = 0x4000000000D and **TreeId** 0x1, and messages have been exchanged such that the current MessageId is 9.

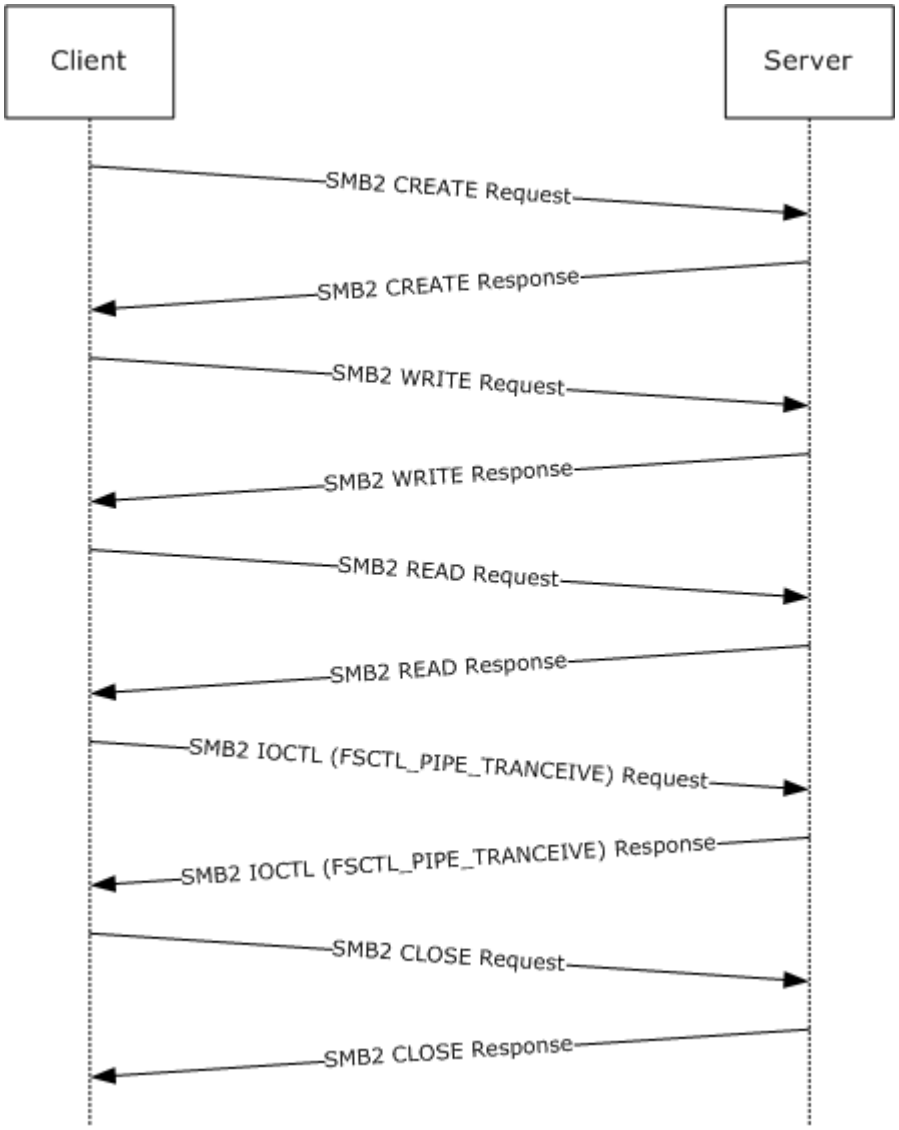


Figure 5: Executing an operation on a named pipe

1. The client sends an SMB2 create request to open the named pipe "srvsvc".

```
Smb2: C CREATE srvsvc
SMB2Header:
Size: 64 (0x40)
Epoch: 0 (0x0)
Status: STATUS_SUCCESS
Command: CREATE
Credits: 111 (0x6F)
Flags: 0 (0x0)
ServerToRedir: ..............................0  Client to Server
AsyncCommand:  ..............................0.  Command is not asynchronous
Related:       ..............................0..  Packet is single message
Signed:        ..............................0...  Packet is not signed
Reserved: 0 (0x0)
```

```
DFS:           0.............................  Command is not a DFS Operation
NextCommand: 0 (0x0)
MessageId: 9 (0x9)
ProcessId: 65279 (0xFEFF)
TreeId: 1 (0x1)
SessionId: 4398046511117 (0x40000000000D)
CCreate:
Size: 57 (0x39)
SecurityFlags: 0 (0x0)
RequestedOplockLevel: 9 (0x9)
ImpersonationLevel: 2 (0x2)
SmbCreateFlags: 0 (0x0)
RootDirectoryFid: 171798757403 (0x280001001B)
DesiredAccess: 0x0012019f
read:        (...............................1) Read Data
write:       (..............................1.) Write Data
append:      (.............................1..) Append Data
readEA:      (............................1...) Read EA
writeEA:     (...........................1....) Write EA
FileExecute: (..........................0.....) No File Execute
FileDeleted: (.........................0......) No File Delete
FileRead:    (........................1.......) File Read Attributes
FileWrite:   (.......................1........) File Write Attributes
FileAttributes: 0x00000000
ReadOnly:    (...............................0) Read/Write
Hidden:      (..............................0.) Not Hidden
System:      (.............................0..) Not System
Reserverd3: 0 (0x0)
Directory:   (...........................0....) File
Archive:     (..........................0.....) Not Archive
Device:      (.........................0......) Not Device
Normal:      (........................0.......) Not Normal
Temporary:   (.......................0........) Permanent
Sparse:      (......................0.........) Not Sparse
Reparse:     (.....................0..........) Not Reparse Point
Compressed:  (....................0...........) Uncompressed
Offline:     (...................0............) Content indexed
NotIndexed:  (..................0.............) Permanent
Encrypted:   (.................0..............) Unencrypted
ShareAccess: Shared for Read/Write
CreateDisposition: Open
CreateOptions: 0x00400040
dir:       (...............................0) non-directory
write:     (..............................0.) non-write through
sq:        (.............................0..) non-sequentially writing allowed
buffer:    (............................0...) intermediate buffering allowed
alert:     (...........................0....) IO alerts bits not set
nonalert:  (..........................0.....) IO non-alerts bit not set
nondir:    (.........................1......) Operation is on non-directory file
connect:   (........................0.......) tree connect bit not set
oplock:    (.......................0........) complete if oplocked bit not set
EA:        (......................0.........) no EA knowledge bit is not set
filename:  (.....................0..........) 8.3 filenames bit is not set
random:    (....................0...........) random access bit is not set
delete:    (...................0............) delete on close bit is not set
open:      (..................0.............) open by filename
backup:    (.................0..............) open for backup bit not set
NameOffset: 120 (0x78)
NameLength: 12 (0xC)
CreateContextsOffset: 0 (0x0)
CreateContextsLength: 0 (0x0)
Name: srvsvc
```

2. The server responds with an SMB2 create response with the FileId for the pipe open.

```
Smb2: R CREATE FID=
SMB2Header:
Size: 64 (0x40)
Epoch: 0 (0x0)
Status: STATUS SUCCESS
Command: CREATE
Credits: 1 (0x1)
Flags: 1 (0x1)
ServerToRedir: ..............................1  Server to Client
AsyncCommand:  ..............................0.  Command is not asynchronous
Related:       ..............................0..  Packet is single message
Signed:        ..............................0...  Packet is not signed
Reserved: 0 (0x0)
DFS:           0.............................  Command is not a DFS Operation
NextCommand: 0 (0x0)
MessageId: 9 (0x9)
ProcessId: 65279 (0xFEFF)
TreeId: 1 (0x1)
SessionId: 4398046511117 (0x40000000000D)
RCreate:
Size: 89 (0x59)
OplockLevel: 0 (0x0)
Reserved1: 9 (0x9)
CreateAction: 1 (0x1)
CreationTime: 0 (0x0)
LastAccessTime: 0 (0x0)
LastWriteTime: 0 (0x0)
LastChangeTime: 0 (0x0)
AllocationSize: 4096 (0x1000)
EndOfFile: 0 (0x0)
FileAttributes: 0x00000080
ReadOnly:   (.............................0) Read/Write
Hidden:     (.............................0.) Not Hidden
System:     (.............................0..) Not System
Reserverd3: 0 (0x0)
Directory:  (...........................0....) File
Archive:    (...........................0.....) Not Archive
Device:     (...........................0......) Not Device
Normal:     (.........................1.......) Normal
Temporary:  (........................0........) Permanent
Sparse:     (.......................0.........) Not Sparse
Reparse:    (......................0..........) Not Reparse Point
Compressed: (.....................0...........) Uncompressed
Offline:    (....................0............) Content indexed
NotIndexed: (...................0.............) Permanent
Encrypted:  (..................0..............) Unencrypted
Reserved2: 7536758 (0x730076)
Fid:
Persistent: 5 (0x5)
Volatile: -4294967291 (0xFFFFFFFF00000005)
CreateContextsOffset: 0 (0x0)
CreateContextsLength: 0 (0x0)
```

3. The client sends an SMB2 write request to write data into the pipe.

```
Smb2: C WRITE 0x74 bytes at offset 0 (0x0)
SMB2Header:
Size: 64 (0x40)
Epoch: 0 (0x0)
```

```
Status: STATUS SUCCESS
Command: WRITE
Credits: 111 (0x6F)
Flags: 0 (0x0)
ServerToRedir: ..............................0  Client to Server
AsyncCommand:  ..............................0.  Command is not asynchronous
Related:       ..............................0..  Packet is single message
Signed:        ..............................0...  Packet is not signed
Reserved: 0 (0x0)
DFS:           0.............................  Command is not a DFS Operation
NextCommand: 0 (0x0)
MessageId: 10 (0xA)
ProcessId: 65279 (0xFEFF)
TreeId: 1 (0x1)
SessionId: 4398046511117 (0x4000000000D)
CWrite:
Size: 49 (0x31)
DataOffset: 112 (0x70)
DataLength: 116 (0x74)
Offset: 0 (0x0)
Fid:
Persistent: 5 (0x5)
Volatile: -4294967291 (0xFFFFFFFF00000005)
Channel: 0 (0x0)
RemainingBytes: 0 (0x0)
WriteChannelInfoOffset: 0 (0x0)
WriteChannelInfoLength: 0 (0x0)
Flags: 0 (0x0)
Data: (116 bytes)
```

4. The server responds with an SMB2 write response indicating the data was written successfully.

```
Smb2: R WRITE 0x74 bytes written
SMB2Header:
Size: 64 (0x40)
Epoch: 0 (0x0)
Status: STATUS_SUCCESS
Command: WRITE
Credits: 1 (0x1)
Flags: 1 (0x1)
ServerToRedir: ..............................1 Server to Client
AsyncCommand:  ..............................0.  Command is not asynchronous
Related:       ..............................0..  Packet is single message
Signed:        ..............................0...  Packet is not signed
Reserved: 0 (0x0)
DFS:           0.............................  Command is not a DFS Operation
NextCommand: 0 (0x0)
MessageId: 10 (0xA)
ProcessId: 65279 (0xFEFF)
TreeId: 1 (0x1)
SessionId: 4398046511117 (0x4000000000D)
RWrite:
Size: 17 (0x11)
Reserved: 0 (0x0)
DataLength: 116 (0x74)
Remaining: 0 (0x0)
WriteChannelInfoOffset: 0 (0x0)
WriteChannelInfoLength: 0 (0x0)
```

5. The client sends an SMB2 read request to read data from the pipe.

```
Smb2: C READ 0x400 bytes from offset 0 (0x0)
SMB2Header:
Size: 64 (0x40)
Epoch: 0 (0x0)
Status: STATUS SUCCESS
Command: READ
Credits: 111 (0x6F)
Flags: 0 (0x0)
ServerToRedir: ..............................0 Client to Server
AsyncCommand:  ..............................0.  Command is not asynchronous
Related:       ..............................0..  Packet is single message
Signed:        ..............................0...  Packet is not signed
Reserved: 0 (0x0)
DFS:           0.............................  Command is not a DFS Operation
NextCommand: 0 (0x0)
MessageId: 11 (0xB)
ProcessId: 65279 (0xFEFF)
TreeId: 1 (0x1)
SessionId: 4398046511117 (0x40000000000D)
CRead:
Size: 49 (0x31)
Padding: 80 (0x50)
Reserved: 0 (0x0)
DataLength: 1024 (0x400)
Offset: 0 (0x0)
Fid:
Persistent: 5 (0x5)
Volatile: -4294967291 (0xFFFFFFFF00000005)
MinimumCount: 0 (0x0)
Channel: 0 (0x0)
RemainingBytes: 0 (0x0)
ReadChannelInfoOffset: 0 (0x0)
ReadChannelInfoLength: 0 (0x0)
```

6. The server responds with an SMB2 read response with the data that was read.

```
Smb2: R READ 0x5c bytes read
SMB2Header:
Size: 64 (0x40)
Epoch: 0 (0x0)
Status: STATUS SUCCESS
Command: READ
Credits: 1 (0x1)
Flags: 1 (0x1)
ServerToRedir: ..............................1 Server to Client
AsyncCommand:  ..............................0.  Command is not asynchronous
Related:       ..............................0..  Packet is single message
Signed:        ..............................0...  Packet is not signed
Reserved: 0 (0x0)
DFS:           0.............................  Command is not a DFS Operation
NextCommand: 0 (0x0)
MessageId: 11 (0xB)
ProcessId: 65279 (0xFEFF)
TreeId: 1 (0x1)
SessionId: 4398046511117 (0x40000000000D)
RRead:
Size: 17 (0x11)
DataOffset: 80 (0x50)
Reserved: 0 (0x0)
```

```
DataLength: 92 (0x5C)
DataRemaining: 0 (0x0)
Reserved2: 0 (0x0)
Data: (92 bytes)
```

7. The client sends an SMB2 IOCTL request to perform a pipe transaction, writing data into the buffer and then reading the response in a single operation.

```
Smb2: C IOCTL
SMB2Header:
Size: 64 (0x40)
Epoch: 0 (0x0)
Status: STATUS_SUCCESS
Command: IOCTL
Credits: 111 (0x6F)
Flags: 0 (0x0)
ServerToRedir: ..............................0 Client to Server
AsyncCommand:  ..............................0.  Command is not asynchronous
Related:       ..............................0..  Packet is single message
Signed:        ..............................0...  Packet is not signed
Reserved: 0 (0x0)
DFS:           0.............................  Command is not a DFS Operation
NextCommand: 0 (0x0)
MessageId: 12 (0xC)
ProcessId: 65279 (0xFEFF)
TreeId: 1 (0x1)
SessionId: 4398046511117 (0x4000000000D)
CIoCtl:
Size: 57 (0x39)
Reserved: 0 (0x0)
Code: 0x0011c017
Fid:
Persistent: 5 (0x5)
Volatile: -4294967291 (0xFFFFFFFF00000005)
InputOffset: 120 (0x78)
InputCount: 68 (0x44)
MaxInputResponse: 0 (0x0)
OutputOffset: 120 (0x78)
OutputCount: 0 (0x0)
MaxOutputResponse: 1024 (0x400)
Flags: 1 (0x1)
Reserved2: 0 (0x0)
Input: (68 bytes)
```

8. The server sends an SMB2 IOCTL response with the data that was read. In this case, the Windows server is also returning the input buffer that was sent but the client ignores it.

```
Smb2: R IOCTL
SMB2Header:
Size: 64 (0x40)
Epoch: 0 (0x0)
Status: STATUS SUCCESS
Command: IOCTL
Credits: 1 (0x1)
Flags: 1 (0x1)
ServerToRedir: ..............................1 Server to Client
```

```
AsyncCommand:  ..............................0.  Command is not asynchronous
Related:       ..............................0..  Packet is single message
Signed:        ..............................0...  Packet is not signed
Reserved: 0 (0x0)
DFS:           0.............................  Command is not a DFS Operation
NextCommand: 0 (0x0)
MessageId: 12 (0xC)
ProcessId: 65279 (0xFEFF)
TreeId: 1 (0x1)
SessionId: 4398046511117 (0x40000000000D)
RIoCtl:
Size: 49 (0x31)
Reserved: 0 (0x0)
Code: 0x0011c017
Method:         ..............................11 Method neither
Function: 0x005
Access:         ................11..............  Read/Write
Device: 0x0011
Fid:
Persistent: 5 (0x5)
Volatile: -4294967291 (0xFFFFFFFF00000005)
InputOffset: 112 (0x70)
InputCount: 68 (0x44)
OutputOffset: 184 (0xB8)
OutputCount: 112 (0x70)
Flags: 0 (0x0)
Reserved2: 0 (0x0)
Input: (68 bytes)
Output: (112 bytes)
```

9. The client sends an SMB2 close request to close the named pipe.

```
Smb2: C CLOSE FID=
SMB2Header:
Size: 64 (0x40)
Epoch: 0 (0x0)
Status: STATUS_SUCCESS
Command: CLOSE
Credits: 111 (0x6F)
Flags: 0 (0x0)
ServerToRedir: ..............................0 Client to Server
AsyncCommand:  ..............................0.  Command is not asynchronous
Related:       ..............................0..  Packet is single message
Signed:        ..............................0...  Packet is not signed
Reserved: 0 (0x0)
DFS:           0.............................  Command is not a DFS Operation
NextCommand: 0 (0x0)
MessageId: 13 (0xD)
ProcessId: 65279 (0xFEFF)
TreeId: 1 (0x1)
SessionId: 4398046511117 (0x40000000000D)
CClose:
Size: 24 (0x18)
Flags: 1 (0x1)
Reserved: 0 (0x0)
Fid:
Persistent: 5 (0x5)
Volatile: -4294967291 (0xFFFFFFFF00000005)
```

10.The server sends an SMB2 close response to indicate the close was successful.

```
Smb2: R CLOSE
SMB2Header:
Size: 64 (0x40)
Epoch: 0 (0x0)
Status: STATUS SUCCESS
Command: CLOSE
Credits: 1 (0x1)
Flags: 1 (0x1)
ServerToRedir: ..............................1 Server to Client
AsyncCommand:  ..............................0.  Command is not asynchronous
Related:       ..............................0..  Packet is single message
Signed:        ..............................0...  Packet is not signed
Reserved: 0 (0x0)
DFS:           0.............................  Command is not a DFS Operation
NextCommand: 0 (0x0)
MessageId: 13 (0xD)
ProcessId: 65279 (0xFEFF)
TreeId: 1 (0x1)
SessionId: 4398046511117 (0x40000000000D)
RClose:
Size: 60 (0x3C)
Flags: 0 (0x0)
Reserved: 0 (0x0)
CreationTime: 0 (0x0)
LastAccessTime: 0 (0x0)
LastWriteTime: 0 (0x0)
LastChangeTime: 0 (0x0)
AllocationSize: 0 (0x0)
EndOfFile: 0 (0x0)
FileAttributes: 0x00000000
ReadOnly:   (.............................0) Read/Write
Hidden:     (.............................0.) Not Hidden
System:     (.............................0..) Not System
Reserverd3: 0 (0x0)
Directory:  (...........................0....) File
Archive:    (..........................0.....) Not Archive
Device:     (.........................0......) Not Device
Normal:     (........................0.......) Not Normal
Temporary:  (.......................0........) Permanent
Sparse:     (......................0.........) Not Sparse
Reparse:    (.....................0..........) Not Reparse Point
Compressed: (....................0...........) Uncompressed
Offline:    (...................0............) Content indexed
NotIndexed: (..................0.............) Permanent
Encrypted:  (.................0..............)
```

4.4 Reading from a Remote File

The following diagram demonstrates the steps taken to open a remote file, read from it, and close it. Assume that this sequence starts on a connection where the session and tree connect have been established as described in previous sections with SessionId of 0x40000000011 and **TreeId** of 0x5, and messages have been exchanged such that the current MessageId is 10.

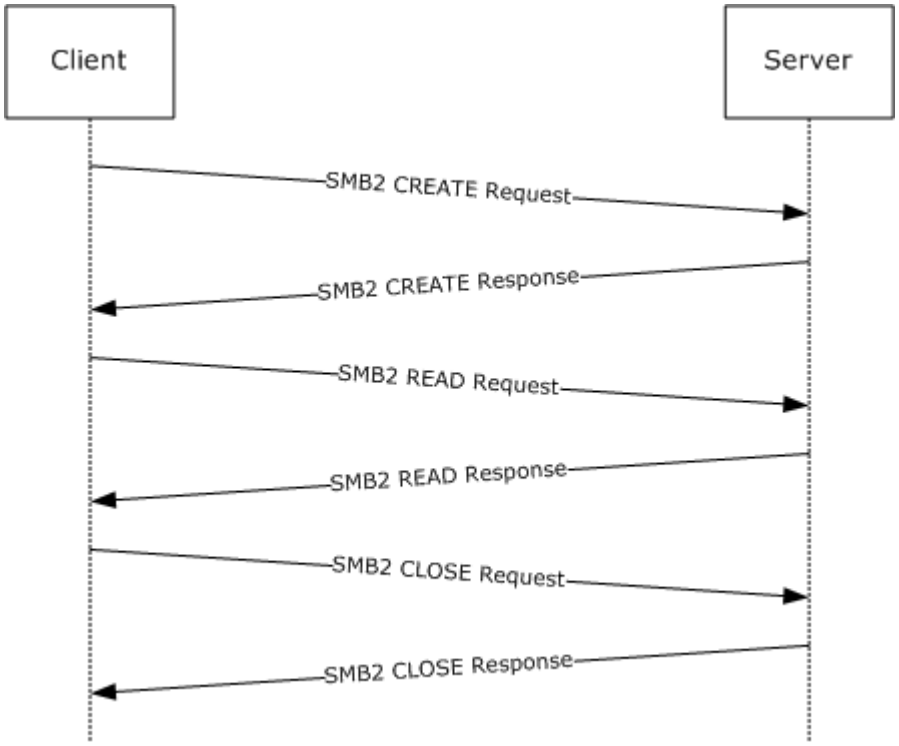


Figure 6: Reading from a remote file

1. The client sends an SMB2 create request for the file "testfile.txt".

```
Smb2: C CREATE testfile.txt
SMB2Header:
Size: 64 (0x40)
Epoch: 0 (0x0)
Status: STATUS_SUCCESS
Command: CREATE
Credits: 111 (0x6F)
Flags: 0 (0x0)
ServerToRedir: ..............................0 Client to Server
AsyncCommand:  ..............................0.  Command is not asynchronous
Related:       ..............................0..  Packet is single message
Signed:        ..............................0...  Packet is not signed
Reserved: 0 (0x0)
DFS:           0.............................  Command is not a DFS Operation
NextCommand: 0 (0x0)
MessageId: 10 (0xA)
ProcessId: 65279 (0xFEFF)
TreeId: 5 (0x5)
SessionId: 4398046511121 (0x40000000011)
CCreate:
Size: 57 (0x39)
SecurityFlags: 0 (0x0)
RequestedOplockLevel: 9 (0x9)
ImpersonationLevel: 2 (0x2)
SmbCreateFlags: 0 (0x0)
RootDirectoryFid: 171798757403 (0x280001001B)
DesiredAccess: 0x00120089
read:        (.............................1) Read Data
write:       (.............................0.) No Write Data
```

```
append:      (.............................0..) No Append Data
readEA:      (............................1...) Read EA
writeEA:     (...........................0....) No Write EA
FileExecute: (..........................0.....) No File Execute
FileDeleted: (.........................0......) No File Delete
FileRead:    (........................1.......) File Read Attributes
FileWrite:   (.......................0........) No File Write Attributes
FileAttributes: 0x00000080
ReadOnly:    (...............................0) Read/Write
Hidden:      (..............................0.) Not Hidden
System:      (.............................0..) Not System
Reserverd3: 0 (0x0)
Directory:   (...........................0....) File
Archive:     (..........................0.....) Not Archive
Device:      (.........................0......) Not Device
Normal:      (........................1.......) Normal
Temporary:   (.......................0........) Permanent
Sparse:      (......................0.........) Not Sparse
Reparse:     (.....................0..........) Not Reparse Point
Compressed:  (....................0...........) Uncompressed
Offline:     (...................0............) Content indexed
NotIndexed:  (..................0.............) Permanent
Encrypted:   (.................0..............) Unencrypted
ShareAccess: Shared for Read/Write
CreateDisposition: Open
CreateOptions: 0x00000060
dir:       (.............................0) non-directory
write:     (............................0.) non-write through
sq:        (...........................0..) non-sequentially writing allowed
buffer:    (..........................0...) intermediate buffering allowed
alert:     (.........................0....) IO alerts bits not set
nonalert:  (........................1.....) Do synchronous IO non-alerts
nondir:    (.......................1......) Operation is on non-directory file
connect:   (......................0.......) tree connect bit not set
oplock:    (.....................0........) complete if oplocked bit not set
EA:        (....................0.........) no EA knowledge bit is not set
filename:  (...................0..........) 8.3 filenames bit is not set
random:    (..................0...........) random access bit is not set
delete:    (.................0............) delete on close bit is not set
open:      (................0.............) open by filename
backup:    (...............0..............) open for backup bit not set
NameOffset: 120 (0x78)
NameLength: 24 (0x18)
CreateContextsOffset: 0 (0x0)
CreateContextsLength: 0 (0x0)
Name: testfile.txt
```

2. The server responds with an SMB2 create response giving the FileId of the opened file.

```
Smb2: R CREATE FID=
SMB2Header:
Size: 64 (0x40)
Epoch: 0 (0x0)
Status: STATUS SUCCESS
Command: CREATE
Credits: 1 (0x1)
Flags: 1 (0x1)
ServerToRedir: ............................1 Server to Client
AsyncCommand:  ............................0.  Command is not asynchronous
Related:       ............................0..  Packet is single message
```

```
Signed:        .........................0...  Packet is not signed
Reserved: 0 (0x0)
DFS:           0.............................  Command is not a DFS Operation
NextCommand: 0 (0x0)
MessageId: 10 (0xA)
ProcessId: 65279 (0xFEFF)
TreeId: 5 (0x5)
SessionId: 4398046511121 (0x40000000011)
RCreate:
Size: 89 (0x59)
OplockLevel: 9 (0x9)
Reserved1: 9 (0x9)
CreateAction: 1 (0x1)
CreationTime: 127972992877715232 (0x1C6A6C24D51DF20)
LastAccessTime: 127972992923579232 (0x1C6A6C2500DB360)
LastWriteTime: 127972992923579232 (0x1C6A6C2500DB360)
LastChangeTime: 127972992923579232 (0x1C6A6C2500DB360)
AllocationSize: 104 (0x68)
EndOfFile: 98 (0x62)
FileAttributes: 0x00000020
ReadOnly:   (.............................0) Read/Write
Hidden:     (.............................0.) Not Hidden
System:     (.............................0..) Not System
Reserverd3: 0 (0x0)
Directory:  (...........................0....) File
Archive:    (...........................1.....) Archive
Device:     (..........................0......) Not Device
Normal:     (.........................0.......) Not Normal
Temporary:  (........................0........) Permanent
Sparse:     (.......................0.........) Not Sparse
Reparse:    (......................0..........) Not Reparse Point
Compressed: (.....................0...........) Uncompressed
Offline:    (....................0............) Content indexed
NotIndexed: (...................0.............) Permanent
Encrypted:  (..................0..............) Unencrypted
Reserved2: 0 (0x0)
Fid:
Persistent: 17 (0x11)
Volatile: -4294967287 (0xFFFFFFFF00000009)
CreateContextsOffset: 0 (0x0)
CreateContextsLength: 0 (0x0)
```

3. The client sends an SMB2 read request to read data from the file.

```
Smb2: C READ 0x62 bytes from offset 0 (0x0)
SMB2Header:
Size: 64 (0x40)
Epoch: 0 (0x0)
Status: STATUS_SUCCESS
Command: READ
Credits: 111 (0x6F)
Flags: 0 (0x0)
ServerToRedir: ..............................0 Client to Server
AsyncCommand:  .............................0.  Command is not asynchronous
Related:       ............................0..  Packet is single message
Signed:        ...........................0...  Packet is not signed
Reserved: 0 (0x0)
DFS:           0.............................  Command is not a DFS Operation
NextCommand: 0 (0x0)
MessageId: 11 (0xB)
ProcessId: 65279 (0xFEFF)
```

```
TreeId: 5 (0x5)
SessionId: 4398046511121 (0x40000000011)
CRead:
Size: 49 (0x31)
Padding: 80 (0x50)
Reserved: 0 (0x0)
DataLength: 98 (0x62)
Offset: 0 (0x0)
Fid:
Persistent: 17 (0x11)
Volatile: -4294967287 (0xFFFFFFFF00000009)
MinimumCount: 0 (0x0)
Channel: 0 (0x0)
RemainingBytes: 0 (0x0)
ReadChannelInfoOffset: 0 (0x0)
ReadChannelInfoLength: 0 (0x0)
```

4. The server responds with an SMB2 read response with the data read from the file.

```
Smb2: R READ 0x62 bytes read
SMB2Header:
Size: 64 (0x40)
Epoch: 0 (0x0)
Status: STATUS SUCCESS
Command: READ
Credits: 1 (0x1)
Flags: 1 (0x1)
ServerToRedir: ..............................1 Server to Client
AsyncCommand:  ..............................0.  Command is not asynchronous
Related:       ..............................0..  Packet is single message
Signed:        ..............................0...  Packet is not signed
Reserved: 0 (0x0)
DFS:           0.............................  Command is not a DFS Operation
NextCommand: 0 (0x0)
MessageId: 11 (0xB)
ProcessId: 65279 (0xFEFF)
TreeId: 5 (0x5)
SessionId: 4398046511121 (0x40000000011)
RRead:
Size: 17 (0x11)
DataOffset: 80 (0x50)
Reserved: 0 (0x0)
DataLength: 98 (0x62)
DataRemaining: 0 (0x0)
Reserved2: 0 (0x0)
Data: (98 bytes)
```

5. The client sends an SMB2 close request to close the file.

```
Smb2: C CLOSE FID=
SMB2Header:
Size: 64 (0x40)
Epoch: 0 (0x0)
Status: STATUS SUCCESS
Command: CLOSE
Credits: 111 (0x6F)
Flags: 0 (0x0)
```

```
ServerToRedir: ..............................0 Client to Server
AsyncCommand:  ..............................0.  Command is not asynchronous
Related:       ..............................0..  Packet is single message
Signed:        ..............................0...  Packet is not signed
Reserved: 0 (0x0)
DFS:           0.............................  Command is not a DFS Operation
NextCommand: 0 (0x0)
MessageId: 12 (0xC)
ProcessId: 65279 (0xFEFF)
TreeId: 5 (0x5)
SessionId: 4398046511121 (0x40000000011)
CClose:
Size: 24 (0x18)
Flags: 1 (0x1) <- Post-query attributes
Reserved: 0 (0x0)
Fid:
Persistent: 9 (0x9)
Volatile: -4294967295 (0xFFFFFFFF00000001)
```

6. The server sends an SMB2 close response indicating the close was successful.

```
Smb2: R CLOSE
SMB2Header:
Size: 64 (0x40)
Epoch: 0 (0x0)
Status: STATUS SUCCESS
Command: CLOSE
Credits: 1 (0x1)
Flags: 1 (0x1)
ServerToRedir: ..............................1 Server to Client
AsyncCommand:  ..............................0.  Command is not asynchronous
Related:       ..............................0..  Packet is single message
Signed:        ..............................0...  Packet is not signed
Reserved: 0 (0x0)
DFS:           0.............................  Command is not a DFS Operation
NextCommand: 0 (0x0)
MessageId: 12 (0xC)
ProcessId: 65279 (0xFEFF)
TreeId: 5 (0x5)
SessionId: 4398046511121 (0x40000000011)
RClose:
Size: 60 (0x3C)
Flags: 1 (0x1)
Reserved: 0 (0x0)
CreationTime: 127972990708847232 (0x1C6A6C1CC0B9280)
LastAccessTime: 127972993090343232 (0x1C6A6C259FE5140)
LastWriteTime: 127972992877715232 (0x1C6A6C24D51DF20)
LastChangeTime: 127972992877715232 (0x1C6A6C24D51DF20)
AllocationSize: 0 (0x0)
EndOfFile: 0 (0x0)
FileAttributes: 0x00000010
ReadOnly:   (...............................0) Read/Write
Hidden:     (...............................0.) Not Hidden
System:     (...............................0..) Not System
Reserverd3: 0 (0x0)
Directory:  (...........................1....) Directory
Archive:    (...........................0.....) Not Archive
Device:     (...........................0......) Not Device
Normal:     (...........................0.......) Not Normal
Temporary:  (.......................0........) Permanent
Sparse:     (......................0.........) Not Sparse
Reparse:    (......................0..........) Not Reparse Point
```

```
Compressed: (.....................0...........) Uncompressed
Offline:    (.....................0...........) Content indexed
NotIndexed: (...................0.............) Permanent
Encrypted:  (...................0.............) Unencrypted
```

4.5 Writing to a Remote File

The following diagram demonstrates the steps taken to open a remote file, write to it, and close it. Assume that this sequence starts on a connection where the session and tree connect have been established as described in previous sections, and messages have been exchanged such that the current MessageId is 30. Let us assume **TreeId** is set to 0x1 and SessionId is set to 0x40000000001 for all requests and responses listed below.

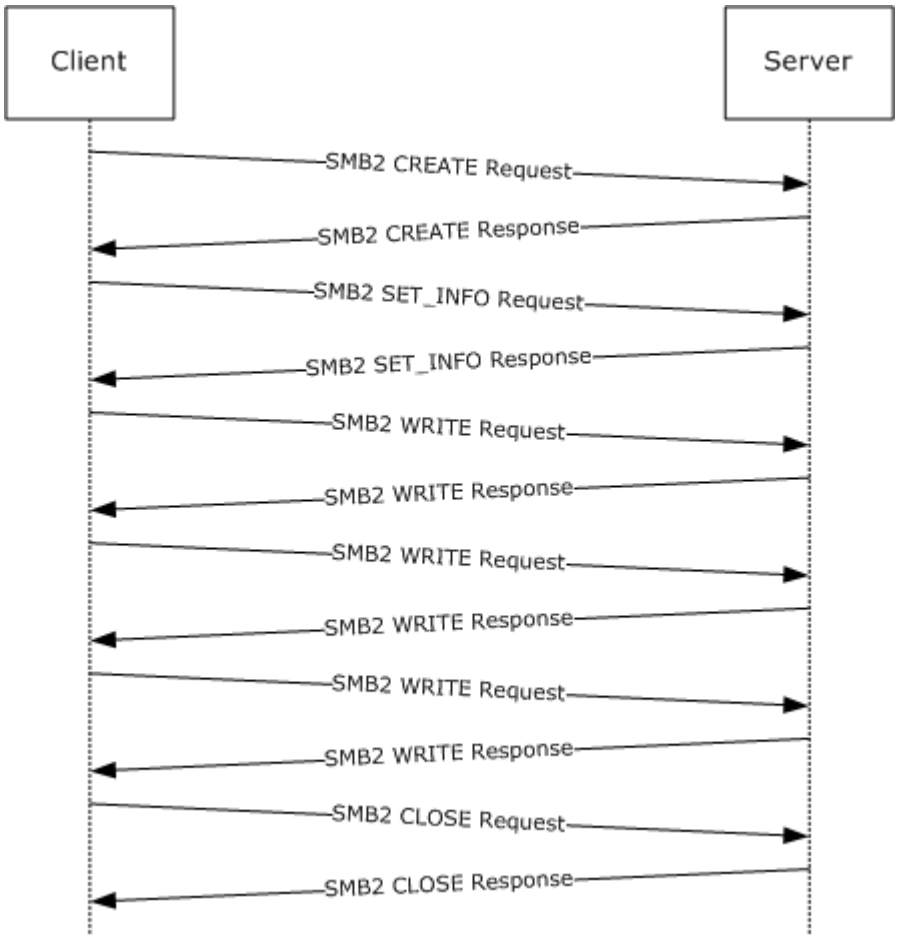


Figure 7: Writing to a remote file

1. The client sends an SMB2 create for the file "test.dat".

```
Smb2: C CREATE test.dat
SMB2Header:
Size: 64 (0x40)
```

```
Epoch: 0 (0x0)
Status: STATUS_SUCCESS
Command: CREATE
Credits: 111 (0x6F)
Flags: 0 (0x0)
ServerToRedir: ..............................0 Client to Server
AsyncCommand:  ..............................0.  Command is not asynchronous
Related:       ..............................0..  Packet is single message
Signed:        ..............................0...  Packet not signed
Reserved: 0 (0x0)
DFS:           0.............................  Command is not a DFS Operation
NextCommand: 0 (0x0)
MessageId: 10 (0xA)
ProcessId: 65279 (0xFEFF)
TreeId: 1 (0x1)
SessionId: 4398046511125 (0x40000000015)
CCreate:
Size: 57 (0x39)
SecurityFlags: 0 (0x0)
RequestedOplockLevel: 9 (0x9)
ImpersonationLevel: 2 (0x2)
SmbCreateFlags: 0 (0x0)
RootDirectoryFid: 171798757403
         (0x280001001B)
DesiredAccess: 0x00130197
read:        (...............................1) Read Data
write:       (...............................1.) Write Data
append:      (...............................1..) Append Data
readEA:      (............................0...) No Read EA
writeEA:     (............................1....) Write EA
FileExecute: (...........................0.....) No File Execute
FileDeleted: (..........................0......) No File Delete
FileRead:    (.........................1.......) File Read Attributes
FileWrite:   (.........................1........) File Write Attributes
FileAttributes: 0x00000020
ReadOnly:    (...............................0) Read/Write
Hidden:      (...............................0.) Not Hidden
System:      (...............................0..) Not System
Reserverd3: 0 (0x0)
Directory:   (............................0....) File
Archive:     (............................1.....) Archive
Device:      (...........................0......) Not Device
Normal:      (..........................0.......) Not Normal
Temporary:   (.........................0........) Permanent
Sparse:      (........................0.........) Not Sparse
Reparse:     (.......................0..........) Not Reparse Point
Compressed:  (......................0...........) Uncompressed
Offline:     (.....................0............) Content indexed
NotIndexed:  (....................0.............) Permanent
Encrypted:   (...................0..............) Unencrypted
ShareAccess: No sharing
CreateDisposition: Overwrite if
CreateOptions: 0x0000004c
dir:         (...............................0) Non-directory
write:       (...............................0.) Non-write through
sq:          (...............................1..) Data must be written
                                                    to the file sequentially
buffer:      (............................1...) Do not do intermediate
                                                    buffering
alert:       (............................0....) IO alerts bits not set
nonalert:    (...........................0.....) IO non-alerts bit not set
nondir:      (..........................1......) Operation is on non-directory
                                                    file
connect:     (.........................0.......) Tree connect bit not set
oplock:      (........................0......
```

```
       ..) Complete if oplocked bit is not
       set
EA:          (.....................0.........) No EA knowledge bit is not set
filename:    (.....................0.........) 8.3 filenames bit is not set
random:      (.....................0.........) Random access bit is not set
delete:      (.....................0.........) Delete on close bit is not set
open:        (.....................0.........) Open by filename
backup:      (.....................0.........) Open for backup bit not set
NameOffset: 120 (0x78)
NameLength: 16 (0x10)
CreateContextsOffset: 0 (0x0)
CreateContextsLength: 0 (0x0)
Name: test.dat
```

2. The server responds with an SMB2 create response with the FileId of the opened file.

```
Smb2: R CREATE FID=
SMBIdentifier: SMB
SMB2Header:
Size: 64 (0x40)
Epoch: 0 (0x0)
Status: STATUS_SUCCESS
Command: CREATE
Credits: 1 (0x1)
Flags: 1 (0x1)
ServerToRedir: ..............................1 Server to Client
AsyncCommand:  ..............................0.  Command not asynchronous
Related:       ..............................0..  Packet is single message
Signed:        ..............................0...  Packet not signed
Reserved: 0 (0x0)
DFS:           0.............................  Command not DFS Operation
NextCommand: 0 (0x0)
MessageId: 10 (0xA)
ProcessId: 65279 (0xFEFF)
TreeId: 1 (0x1)
SessionId: 4398046511125 (0x40000000015)
RCreate:
Size: 89 (0x59)
OplockLevel: 9 (0x9)
Reserved1: 9 (0x9)
CreateAction: 2 (0x2)
CreationTime: 127972994486543232 (0x1C6A6C2AD36A380)
LastAccessTime: 127972994486543232 (0x1C6A6C2AD36A380)
LastWriteTime: 127972994486543232 (0x1C6A6C2AD36A380)
LastChangeTime: 127972994486543232 (0x1C6A6C2AD36A380)
AllocationSize: 765952 (0xBB000)
EndOfFile: 0 (0x0)
FileAttributes: 0x00000020
ReadOnly:    (.............................0) Read/Write
Hidden:      (.............................0.) Not Hidden
System:      (.............................0..) Not System
Reserverd3: 0 (0x0)
Directory:   (...........................0....) File
Archive:     (...........................1.....) Archive
Device:      (.........................0......) Not Device
Normal:      (.........................0.......) Not Normal
Temporary:   (.......................0........) Permanent
Sparse:      (.....................0..........) Not Sparse
Reparse:     (.....................0..........) Not Reparse Point
Compressed:  (...................0............) Uncompressed
Offline:     (...................0............) Content indexed
NotIndexed:  (.................0..............) Permanent
```

```
Encrypted:  (.................0.............) Unencrypted
Reserved2: 0 (0x0)
Fid:
Persistent: 25 (0x19)
Volatile: -4294967291
         (0xFFFFFFFF00000005)
CreateContextsOffset: 0 (0x0)
CreateContextsLength: 0 (0x0)
```

3. The client sends an SMB2 set info request to set FileEndOfFileInformation to 0x2f000.

```
Smb2: C SET INFORMATION
SMB2Header:
Size: 64 (0x40)
Epoch: 0 (0x0)
Status: STATUS SUCCESS
Command: SET INFORMATION
Credits: 111 (0x6F)
Flags: 0 (0x0)
ServerToRedir: ..............................0 Client to Server
AsyncCommand:  ..............................0.  Command not asynchronous
Related:       ..............................0..  Packet is single message
Signed:        ..............................0...  Packet not signed
Reserved: 0 (0x0)
DFS:           0.............................  Command not DFS Operation
NextCommand: 0 (0x0)
MessageId: 11 (0xB)
ProcessId: 65279 (0xFEFF)
TreeId: 1 (0x1)
SessionId: 4398046511125 (0x40000000015)
CSetInfo:
Size: 33 (0x21)
InfoType: 1 (0x1)
FileInformationClass:
         FileEndOfFileInformation
BufferLength: 8 (0x8)
BufferOffset: 96 (0x60)
Reserved: 0 (0x0)
SecurityInformation: 0 (0x0)
Fid:
Persistent: 25 (0x19)
Volatile: -4294967291
         (0xFFFFFFFF00000005)
Buffer: (8 bytes) 0x000000000002f000
```

4. The server sends an SMB2 set info response with success.

```
Smb2: R SET INFORMATION
SMB2Header:
Size: 64 (0x40)
Epoch: 0 (0x0)
Status: STATUS_SUCCESS
Command: SET INFORMATION
Credits: 1 (0x1)
Flags: 1 (0x1)
ServerToRedir: ..............................1 Server to Client
AsyncCommand:  ..............................0.  Command not asynchronous
Related:       ..............................0..  Packet is single message
Signed:        ..............................0...  Packet not signed
```

```
Reserved: 0 (0x0)
DFS:           0.............................  Command not DFS Operation
NextCommand: 0 (0x0)
MessageId: 11 (0xB)
ProcessId: 65279 (0xFEFF)
TreeId: 1 (0x1)
SessionId: 4398046511125 (0x40000000015)
RSetInfo:
Size: 2 (0x2)
```

5. The client sends an SMB2 write request to write the first 0x10000 bytes.

```
Smb2: C WRITE 0x10000 bytes at
         offset 0 (0x0)
SMB2Header:
Size: 64 (0x40)
Epoch: 0 (0x0)
Status: STATUS SUCCESS
Command: WRITE
Credits: 111 (0x6F)
Flags: 0 (0x0)
ServerToRedir: ..............................0 Client to Server
AsyncCommand:  ..............................0.  Command not asynchronous
Related:       ..............................0..  Packet is single message
Signed:        ..............................0...  Packet not signed
Reserved: 0 (0x0)
DFS:           0.............................  Command not DFS Operation
NextCommand: 0 (0x0)
MessageId: 12 (0xC)
ProcessId: 65279 (0xFEFF)
TreeId: 1 (0x1)
SessionId: 4398046511125 (0x40000000015)
CWrite:
Size: 49 (0x31)
DataOffset: 112 (0x70)
DataLength: 65536 (0x10000)
Offset: 0 (0x0)
Fid:
Persistent: 25 (0x19)
Volatile: -4294967291
         (0xFFFFFFFF00000005)
Channel: 0 (0x0)
RemainingBytes: 0 (0x0)
WriteChannelInfoOffset: 0 (0x0)
WriteChannelInfoLength: 0 (0x0)
Flags: 0 (0x0)
```

6. The server responds with an SMB2 write response indicating 0x10000 bytes were written.

```
Smb2: R WRITE 0x10000 bytes
         written
  SMB2Header:
         Size: 64 (0x40)
         Epoch: 0 (0x0)
         Status: STATUS SUCCESS
         Command: WRITE
         Credits: 1 (0x1)
         Flags: 1 (0x1)
               ServerToRedir: ..............................1 Server to Client
```

```
               AsyncCommand:  ..............................0.  Command not
asynchronous
               Related:       ..............................0..  Packet is single
message
               Signed:        ..............................0...  Packet not signed
               Reserved: 0 (0x0)
               DFS:           0.............................  Command not DFS
Operation
         NextCommand: 0 (0x0)
         MessageId: 12 (0xC)
         ProcessId: 65279 (0xFEFF)
         TreeId: 1 (0x1)
         SessionId: 4398046511125 (0x40000000015)
  RWrite:
         Size: 17 (0x11)
         Reserved: 0 (0x0)
         DataLength: 65536 (0x10000)
         Remaining: 0 (0x0)
         WriteChannelInfoOffset: 0 (0x0)
         WriteChannelInfoLength: 0 (0x0)
```

7. The client sends an SMB2 write request to write the next 0x10000 bytes.

```
Smb2: C WRITE 0x10000 bytes at
         offset 0 (0x0)
SMB2Header:
Size: 64 (0x40)
Epoch: 0 (0x0)
Status: STATUS SUCCESS
Command: WRITE
Credits: 111 (0x6F)
Flags: 0 (0x0)
ServerToRedir: ..............................0 Client to Server
AsyncCommand:  ..............................0.  Command not asynchronous
Related:       ..............................0..  Packet is single message
Signed:        ..............................0...  Packet not signed
Reserved: 0 (0x0)
DFS:           0.............................  Command not DFS Operation
NextCommand: 0 (0x0)
MessageId: 13 (0xD)
ProcessId: 65279 (0xFEFF)
TreeId: 1 (0x1)
SessionId: 4398046511125 (0x40000000015)
CWrite:
Size: 49 (0x31)
DataOffset: 112 (0x70)
DataLength: 65536 (0x10000)
Offset: 65536 (0x10000)
Fid:
Persistent: 25 (0x19)
Volatile: -4294967291
         (0xFFFFFFFF00000005)
Channel: 0 (0x0)
RemainingBytes: 0 (0x0)
WriteChannelInfoOffset: 0 (0x0)
WriteChannelInfoLength: 0 (0x0)
Flags: 0 (0x0)
```

8. The server responds with an SMB2 write response indicating 0x10000 bytes were written.

```
Smb2: R WRITE 0x10000 bytes
        written
SMB2Header:
Size: 64 (0x40)
Epoch: 0 (0x0)
Status: STATUS SUCCESS
Command: WRITE
Credits: 1 (0x1)
Flags: 1 (0x1)
ServerToRedir: ..............................1 Server to Client
AsyncCommand:  ..............................0.  Command not asynchronous
Related:       ..............................0..  Packet is single message
Signed:        ..............................0...  Packet not signed
Reserved: 0 (0x0)
DFS:           0.............................  Command not DFS Operation
NextCommand: 0 (0x0)
MessageId: 13 (0xD)
ProcessId: 65279 (0xFEFF)
TreeId: 1 (0x1)
SessionId: 4398046511125 (0x40000000015)
RWrite:
Size: 17 (0x11)
Reserved: 0 (0x0)
DataLength: 65536 (0x10000)
Remaining: 0 (0x0)
WriteChannelInfoOffset: 0 (0x0)
WriteChannelInfoLength: 0 (0x0)
```

9. The client sends an SMB2 write request to write the final 0xf000 bytes.

```
Smb2: C WRITE 0xF000 bytes at
        offset 0 (0x0)
SMB2Header:
Size: 64 (0x40)
Epoch: 0 (0x0)
Status: STATUS SUCCESS
Command: WRITE
Credits: 111 (0x6F)
Flags: 0 (0x0)
ServerToRedir: ..............................0 Client to Server
AsyncCommand:  ..............................0.  Command not asynchronous
Related:       ..............................0..  Packet is single message
Signed:        ..............................0...  Packet is not signed
Reserved: 0 (0x0)
DFS:           0.............................  Command not DFS Operation
NextCommand: 0 (0x0)
MessageId: 14 (0xE)
ProcessId: 65279 (0xFEFF)
TreeId: 1 (0x1)
SessionId: 4398046511125 (0x40000000015)
CWrite:
Size: 49 (0x31)
DataOffset: 112 (0x70)
DataLength: 61440 (0xF000)
Offset: 131072 (0x20000)
Fid:
Persistent: 25 (0x19)
Volatile: -4294967291
        (0xFFFFFFFF00000005)
Channel: 0 (0x0)
RemainingBytes: 0 (0x0)
WriteChannelInfoOffset: 0 (0x0)
```

```
WriteChannelInfoLength: 0 (0x0)
Flags: 0 (0x0)
```

10.The server responds with an SMB2 write response indicating 0xf000 bytes were written.

```
Smb2: R WRITE 0xF000 bytes
        written
SMB2Header:
Size: 64 (0x40)
Epoch: 0 (0x0)
Status: STATUS SUCCESS
Command: WRITE
Credits: 1 (0x1)
Flags: 1 (0x1)
ServerToRedir: ..............................1 Server to Client
AsyncCommand:  ..............................0.  Command not asynchronous
Related:       ..............................0..  Packet is single message
Signed:        ..............................0...  Packet not signed
Reserved: 0 (0x0)
DFS:           0.............................  Command not DFS Operation
NextCommand: 0 (0x0)
MessageId: 14 (0xE)
ProcessId: 65279 (0xFEFF)
TreeId: 1 (0x1)
SessionId: 4398046511125 (0x40000000015)
RWrite:
Size: 17 (0x11)
Reserved: 0 (0x0)
DataLength: 61440 (0xF000)
Remaining: 0 (0x0)
WriteChannelInfoOffset: 0 (0x0)
WriteChannelInfoLength: 0 (0x0)
```

11.The client sends an SMB2 close request to close the opened file.

```
Smb2: C CLOSE FID=
SMB2Header:
Size: 64 (0x40)
Epoch: 0 (0x0)
Status: STATUS_SUCCESS
Command: CLOSE
Credits: 111 (0x6F)
Flags: 0 (0x0)
ServerToRedir: ..............................0 Client to Server
AsyncCommand:  ..............................0.  Command not asynchronous
Related:       ..............................0..  Packet is single message
Signed:        ..............................0...  Packet not signed
Reserved: 0 (0x0)
DFS:           0.............................  Command not DFS Operation
NextCommand: 0 (0x0)
MessageId: 15 (0xF)
ProcessId: 65279 (0xFEFF)
TreeId: 1 (0x1)
SessionId: 4398046511125 (0x40000000015)
CClose:
Size: 24 (0x18)
Flags: 1 (0x1)
Reserved: 0 (0x0)
Fid:
```

```
Persistent: 25 (0x19)
Volatile: -4294967291
         (0xFFFFFFFF00000005)
```

12.The server sends an SMB2 close response indicating the close was successful.

```
Smb2: R CLOSE
SMBIdentifier: SMB
SMB2Header:
Size: 64 (0x40)
Epoch: 0 (0x0)
Status: STATUS SUCCESS
Command: CLOSE
Credits: 1 (0x1)
Flags: 1 (0x1)
ServerToRedir: ..............................1 Server to Client
AsyncCommand:  ..............................0.  Command not asynchronous
Related:       ..............................0..  Packet is single message
Signed:        ..............................0...  Packet not signed
Reserved: 0 (0x0)
DFS:           0.............................  Command not DFS Operation
NextCommand: 0 (0x0)
MessageId: 15 (0xF)
ProcessId: 65279 (0xFEFF)
TreeId: 1 (0x1)
SessionId: 4398046511125 (0x40000000015)
RClose:
Size: 60 (0x3C)
Flags: 1 (0x1)
Reserved: 0 (0x0)
CreationTime: 127972994486543232
         (0x1C6A6C2AD36A380)
LastAccessTime: 127972994494343232
         (0x1C6A6C2ADADA840)
LastWriteTime: 127965940833141721
         (0x1C6A0585EB543D9)
LastChangeTime: 127972993511484705
         (0x1C6A6C273186D21)
AllocationSize: 196608 (0x30000)
EndOfFile: 192512 (0x2F000)
FileAttributes: 0x00000020
ReadOnly:   (.............................0) Read/Write
Hidden:     (.............................0.) Not Hidden
System:     (.............................0..) Not System
Reserverd3: 0 (0x0)
Directory:  (...........................0....) File
Archive:    (...........................1.....) Archive
Device:     (..........................0......) Not Device
Normal:     (.........................0.......) Not Normal
Temporary:  (........................0........) Permanent
Sparse:     (.......................0.........) Not Sparse
Reparse:    (......................0..........) Not Reparse Point
Compressed: (.....................0...........) Uncompressed
Offline:    (....................0............) Content indexed
NotIndexed: (...................0.............) Permanent
Encrypted:  (..................0..............) Unencrypted
```

4.6 Disconnecting a Share and Logging Off

The following diagram demonstrates the steps taken to close a tree connect and log off a session. Assume that this sequence starts on a connection where the session and tree connect have been established as described in previous sections with SessionId of 0x40000000015 and **TreeId** of 0x1.

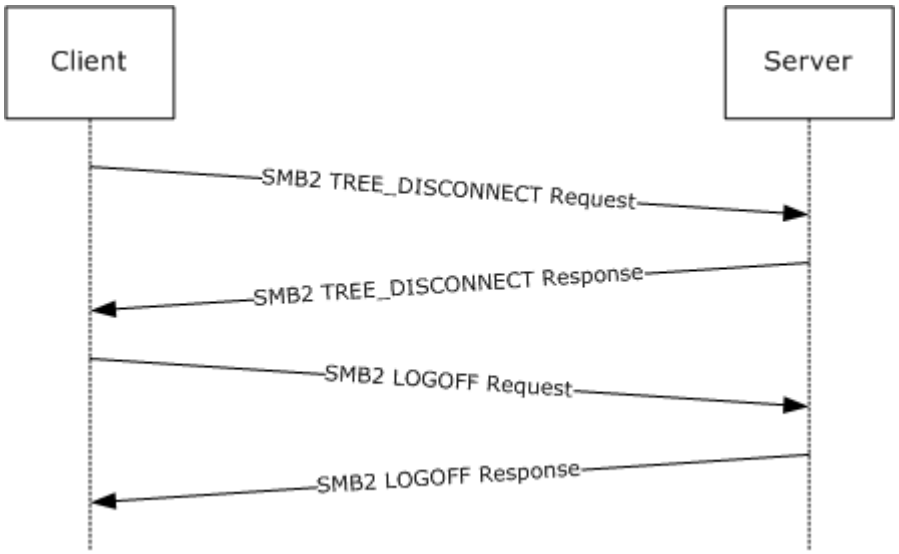


Figure 8: Disconnecting a share and logging off a session

1. The client sends an SMB2 tree disconnect request for the tree connect.

```
Smb2: C TREE DISCONNECT TID=0x1
SMB2Header:
Size: 64 (0x40)
Epoch: 0 (0x0)
Status: STATUS SUCCESS
Command: TREE DISCONNECT
Credits: 111 (0x6F)
Flags: 0 (0x0)
ServerToRedir: ..............................0 Client to Server
AsyncCommand:  ..............................0.  Command is not asynchronous
Related:       ..............................0..  Packet is single message
Signed:        ..............................0...  Packet is not signed
Reserved: 0 (0x0)
DFS:           0.............................  Command is not a DFS Operation
NextCommand: 0 (0x0)
MessageId: 32 (0x20)
ProcessId: 65279 (0xFEFF)
TreeId: 1 (0x1)
SessionId: 4398046511125 (0x40000000015)
CTreeDisconnect:
Size: 4 (0x4)
Reserved: 0 (0x0)
```

2. The server responds with an SMB2 tree disconnect response indicating success.

```
Smb2: R TREE DISCONNECT
```

```
SMB2Header:
Size: 64 (0x40)
Epoch: 0 (0x0)
Status: STATUS SUCCESS
Command: TREE DISCONNECT
Credits: 1 (0x1)
Flags: 1 (0x1)
ServerToRedir: ..............................1 Server to Client
AsyncCommand:  ..............................0.  Command is not asynchronous
Related:       ..............................0..  Packet is single message
Signed:        ..............................0...  Packet is not signed
Reserved: 0 (0x0)
DFS:           0.............................  Command is not a DFS Operation
NextCommand: 0 (0x0)
MessageId: 32 (0x20)
ProcessId: 65279 (0xFEFF)
TreeId: 1 (0x1)
SessionId: 4398046511125 (0x40000000015)
RTreeDisconnect:
Size: 4 (0x4)
Reserved: 0 (0x0)
```

3. The client sends an SMB2 logoff for the session.

```
Smb2: C LOGOFF
SMB2Header:
Size: 64 (0x40)
Epoch: 0 (0x0)
Status: STATUS SUCCESS
Command: LOGOFF
Credits: 111 (0x6F)
Flags: 0 (0x0)
ServerToRedir: ..............................0 Client to Server
AsyncCommand:  ..............................0.  Command is not asynchronous
Related:       ..............................0..  Packet is single message
Signed:        ..............................0...  Packet is not signed
Reserved: 0 (0x0)
DFS:           0.............................  Command is not a DFS Operation
NextCommand: 0 (0x0)
MessageId: 33 (0x21)
ProcessId: 65279 (0xFEFF)
TreeId: 0 (0x0)
SessionId: 4398046511125 (0x40000000015)
CLogoff:
Size: 4 (0x4)
Reserved: 0 (0x0)
```

4. The server responds with an SMB2 logoff response indicating success.

```
Smb2: R LOGOFF
SMB2Header:
Size: 64 (0x40)
Epoch: 0 (0x0)
Status: STATUS SUCCESS
Command: LOGOFF
Credits: 1 (0x1)
Flags: 1 (0x1)
```

```
ServerToRedir: ..............................1 Server to Client
AsyncCommand:  ..............................0.  Command is not asynchronous
Related:       ..............................0..  Packet is single message
Signed:        ..............................0...  Packet is not signed
Reserved: 0 (0x0)
DFS:           0.............................  Command is not a DFS Operation
NextCommand: 0 (0x0)
MessageId: 33 (0x21)
ProcessId: 65279 (0xFEFF)
TreeId: 0 (0x0)
SessionId: 4398046511125 (0x40000000015)
RLogoff:
Size: 4 (0x4)
Reserved: 0 (0x0)
```

5 Security

The following sections specify security considerations for implementers of the Server Message Block (SMB) Version 2.0 Protocol.

5.1 Security Considerations for Implementers

The protocol does not sign oplock break requests from the server to the client if message signing is enabled. This could allow attackers to affect performance but it does not allow them to deny access or alter data.

The previous versions support does potentially allow access to versions of a file that have been deleted or modified, and so could allow access to information that was not available without these extensions. However, this access is still subject to the same access checks it would have normally been subject to.

All data exchanged on the wire is not encrypted. For data that requires stricter security, either the underlying transport must provide encryption of the data or a different protocol may be more applicable.

5.2 Index of Security Parameters

Security parameter	Section
SHA-256 hashing	3.1.4.1
SHA-256 hashing	3.1.5.1
GSSAPI authentication	3.2.4.2.3
GSSAPI authentication	3.3.5.5.3

6 Appendix A: Windows Behavior

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

- Windows Vista
- 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.

<1> Section 1.6: The Server Message Block (SMB) Version 2.0 Protocol is only available in Windows Vista–based clients and servers. Windows Vista–based clients and servers default to using the SMB 2.0 Protocol when it is available. All other releases only support the SMB Protocol, as specified in [MS-SMB], and thus use it by default.

<2> Section 2.2.2: Windows based SMB servers leave this one byte of ErrorData uninitialized and it might contain any value.

<3> Section 2.2.2.1: Windows Server 2003, Windows Vista, and Windows Server 2008 do not zero out the Flags field.

<4> Section 2.2.2.1: Windows Vista and Windows Server 2008 will return the an absolute target to a local resource in the format of \??\C:\... where C: is the drive mount point on the local system and ... is replaced by the remainder of the path to the target.

<5> Section 2.2.3: A Windows Vista RTM–based client would send a value of zero in Dialects array in SMB2 NEGOTIATE request and a Windows Vista RTM based server would acknowledge with a value of 6 in DialectRevision in SMB2 NEGOTIATE Response. This behavior is deprecated.

<6> Section 2.2.4: A Windows Vista RTM–based client would send a value of zero in Dialects array in SMB2 NEGOTIATE request and a Windows Vista RTM based server would acknowledge with a value of 6 in DialectRevision in SMB2 NEGOTIATE Response. This behavior is deprecated.

<7> Section 2.2.4: Windows-based SMB2 servers generate a new ServerGuid each time they are started.

<8> Section 2.2.4: Windows clients do not enforce the MaxTransactSize value.

<9> Section 2.2.5: Windows clients set this bit to any value.

<10> Section 2.2.5: Windows clients set this bit to any value.

<11> Section 2.2.5: Windows clients set this bit to any value.

<12> Section 2.2.9: The Windows SMB 2.0 Protocol client translates any names of the form \\server\pipe to \\server\IPC$ before sending a request on the network.

<13> Section 2.2.13: Windows-based clients never use exclusive oplocks. Because there are no situations where the client would want an exclusive oplock where it would not also want an SMB2_OPLOCK_LEVEL_BATCH, it always requests an SMB2_OPLOCK_LEVEL_BATCH.

<14> Section 2.2.13: Windows clients set this field to any value, and Windows-based servers ignore this value.

<15> Section 2.2.13: Windows-based servers ignore these ShareAccess values.

<16> Section 2.2.13: Windows-based servers ignore this value.

<17> Section 2.2.13: Windows-based servers ignore this value.

<18> Section 2.2.13: Windows-based servers ignore this value.

<19> Section 2.2.13.2.3: Windows-based SMB2 Server implementations allow a durable open only on an SMB2 CREATE with a RequestedOplockLevel of SMB2_OPLOCK_LEVEL_BATCH.

<20> Section 2.2.14: Windows-based clients never use exclusive oplocks. Because there are no situations where it would want an exclusive oplock where it would not also want an SMB2_OPLOCK_LEVEL_BATCH, it always requests an SMB2_OPLOCK_LEVEL_BATCH.

<21> Section 2.2.14.2.4: Windows servers send an SMB2 CREATE_CONTEXT response as specified in section 2.2.13.2.3.

<22> Section 2.2.23: Windows-based clients never use exclusive oplocks. Because there are no situations where it would want an exclusive oplock where it would not also want an SMB2_OPLOCK_LEVEL_BATCH, it always requests an SMB2_OPLOCK_LEVEL_BATCH.

<23> Section 2.2.24: Windows-based clients never use exclusive oplocks. There are no situations where an exclusive oplock would be used instead of using a SMB2_OPLOCK_LEVEL_BATCH.

<24> Section 2.2.25: Windows-based clients never use exclusive oplocks. There are no situations where an exclusive oplock would be used instead of using a SMB2_OPLOCK_LEVEL_BATCH.

<25> Section 2.2.31: Windows-based clients set this field to any value.

<26> Section 2.2.31: Windows-based clients set this field to any value.

<27> Section 2.2.32: Windows clients set the InputOffset field to the offset, in bytes, from the beginning of the SMB2 header to the Buffer[] field of the response.

<28> Section 2.2.32.3: Windows Server 2003, Windows Vista and Windows Server 2008 do not zero out the extra bytes of the Context field.

<29> Section 2.2.33: Windows-based servers do not support this value.

<30> Section 2.2.37.1: Windows-based clients will always request quota information by using a SidList, which is a linked list of one or more FILE_GET_QUOTA_INFORMATION structures.

<31> Section 3.1.3: By default, Windows-based clients and servers set the RequireMessageSigning value to TRUE for domain controllers and FALSE for all other machines.

<32> Section 3.1.5.1: Windows SMB2 servers return the error STATUS_ACCESS_DENIED for incorrectly signed packets. Windows SMB2 clients will discard any incorrectly signed packets.

<33> Section 3.2.1.3: Windows clients do not enforce the MaxTransactSize value.

<34> Section 3.2.2.1: The Windows-based client implements this timer with a default value of 60 seconds.

<35> Section 3.2.2.2: The Windows-based client uses this timer with a default time-out value of 10 seconds.

<36> Section 3.2.4.1.2: The Windows-based client will request credits up to a configured maximum. That maximum is 128 by default.

<37> Section 3.2.4.1.4: Windows SMB2 Server allows a mix of related and unrelated compound requests in the same transport send. Upon encountering a request with SMB2_FLAGS_RELATED_OPERATIONS not set Windows SMB2 Server treats it as start of a chain.

<38> Section 3.2.4.1.4: Windows Vista and Windows Server 2008 will align their compounded requests and responses on 8 byte boundaries. Currently they do not disconnect other machines that disobey this rule.

<39> Section 3.2.4.1.4: The Windows-based client does not send unrelated compounded requests.

<40> Section 3.2.4.1.4: Windows-based clients will compound certain related requests to improve performance, usually combining a Create with another operation, such as an attribute query.

<41> Section 3.2.4.2.1: Windows clients always re-use an existing connection, if any, to the Windows server.

<42> Section 3.2.4.2.2: The Windows-based client will initiate a multi-protocol negotiation unless it has previously negotiated with this server and determined that it supports the SMB 2.0 Protocol. In the latter case, it will initiate an SMB2-only negotiate.

<43> Section 3.2.4.2.2.1: When a Windows based client sends the deprecated "SMB2.001" dialect, a Windows Vista RTM based server would acknowledge with a value of 6 in DialectRevision in SMB2 NEGOTIATE Response. This behavior is deprecated.

<44> Section 3.2.4.2.2.2: Windows-based clients set Capabilities, ClientGuid, and ClientStartTime to 0.

<45> Section 3.2.4.2.3: Windows-based clients implement the first option that is specified.

<46> Section 3.2.4.2.3: All the GSS-API tokens used by windows SMB clients are up to 4Kbytes in size. SMB servers always instruct the GSS_API server to expect the GSS_C_FRAGMENT_TO_FIT.

<47> Section 3.2.4.2.3: Windows SMB2 Client does not set the SMB2_NEGOTIATE_SIGNING_ENABLED bit if RequireMessageSigning is TRUE.

<48> Section 3.2.4.2.3.1: Windows-based clients implement the first option that is specified.

<49> Section 3.2.4.2.3.1: All the GSS-API tokens used by windows SMB clients are up to 4Kbytes in size. SMB servers always instruct the GSS_API server to expect the GSS_C_FRAGMENT_TO_FIT.

<50> Section 3.2.4.2.3.1: Windows SMB2 Client does not set the SMB2_NEGOTIATE_SIGNING_ENABLED bit if RequireMessageSigning is TRUE.

<51> Section 3.2.4.3: Windows-based clients will request a batch oplock for file creates.

<52> Section 3.2.4.4: Windows-based clients will attempt to reconnect to an open if durability was granted at open time.

<53> Section 3.2.4.5: Windows-based clients will attempt to reconnect to opens if durability was granted at open time.

<54> Section 3.2.4.5: Windows-based clients will try to send multiple read commands at the same time, starting at the lowest offset and working to the highest.

<55> Section 3.2.4.6: Windows-based clients will attempt to reconnect to an open if durability was granted at open time.

<56> Section 3.2.4.6: Windows-based clients always put the payload at the beginning of the **Buffer** field and do not insert padding.

<57> Section 3.2.4.6: Windows-based clients will try to send multiple write commands at the same time, starting at the lowest offset and working to the highest.

<58> Section 3.2.4.7: Windows clients will attempt to reconnect to an open if durability was granted at open time.

<59> Section 3.2.4.8: Windows-based clients will attempt to reconnect to an open if durability was granted at open time.

<60> Section 3.2.4.9: Windows-based clients will attempt to reconnect to an open if durability was granted at open time.

<61> Section 3.2.4.10: Windows-based clients will attempt to reconnect to an open if durability was granted at open time.

<62> Section 3.2.4.11: Windows-based clients will attempt to reconnect to an open if durability was granted at open time.

<63> Section 3.2.4.12: Windows-based clients will attempt to reconnect to an open if durability was granted at open time.

<64> Section 3.2.4.13: Windows-based clients will attempt to reconnect to an open if durability was granted at open time.

<65> Section 3.2.4.14: Windows-based clients will attempt to reconnect to an open if durability was granted at open time.

<66> Section 3.2.4.15: Windows-based clients will attempt to reconnect to an open if durability was granted at open time.

<67> Section 3.2.4.16: Windows-based clients will attempt to reconnect to an open if durability was granted at open time.

<68> Section 3.2.4.17: Windows-based clients will attempt to reconnect to an open if durability was granted at the open time.

<69> Section 3.2.4.18: Windows-based clients will attempt to reconnect to an open if durability was granted at open time.

<70> Section 3.2.4.19.1: Windows-based clients will attempt to reconnect to an open if durability was granted at open time.

<71> Section 3.2.4.19.1: Windows clients set the OutputOffset field to InputOffset + InputCount, rounded up to a multiple of 8 bytes.

<72> Section 3.2.4.19.2: Windows-based clients will attempt to reconnect to an open if durability was granted at open time.

<73> Section 3.2.4.19.2: Windows-based clients will attempt to reconnect to an open if durability was granted at open time.

<74> Section 3.2.4.19.2: Windows clients set the **OutputOffset** field to InputOffset + InputCount, rounded up to a multiple of 8 bytes.

<75> Section 3.2.4.19.3: Windows clients set the OutputOffset field to InputOffset + InputCount, rounded up to a multiple of 8 bytes.

<76> Section 3.2.4.19.4: Windows-based clients will attempt to reconnect to an open if durability was granted at open time.

<77> Section 3.2.4.19.4: Windows clients set the **OutputOffset** field to InputOffset + InputCount, rounded up to a multiple of 8 bytes.

<78> Section 3.2.4.19.5: Windows-based clients will attempt to reconnect to an open if durability was granted at open time.

<79> Section 3.2.4.19.5: Windows clients set the OutputOffset field to InputOffset + InputCount, rounded up to a multiple of 8 bytes.

<80> Section 3.2.4.19.6: Windows-based SMB2 servers do not support IOCTL requests.

<81> Section 3.2.4.19.6: Windows-based clients will attempt to reconnect to an open if durability was granted at open time.

<82> Section 3.2.4.19.6: Windows clients set the OutputOffset field to InputOffset + InputCount, rounded up to a multiple of 8 bytes.

<83> Section 3.2.4.20: Windows-based clients will attempt to reconnect to an open if durability was granted at open time.

<84> Section 3.2.5.1.2: Windows-based clients will not disconnect the connection but simply disregard the incorrectly signed response.

<85> Section 3.2.5.1.6: Windows-based clients will not disconnect the connection, but will simply fail the request.

<86> Section 3.2.5.1.7: Windows-based SMB 2.0 Protocol clients do not check the validity of the command in the response.

<87> Section 3.2.5.2: Windows-based clients will not use the MaxTransactSize and will use the ServerGuid to determine if the client and server are the same machine.

<88> Section 3.2.5.19.1: Windows-based clients will not request exclusive oplocks.

<89> Section 3.2.6.1: The Windows-based client will fail requests for which no reply is received within the default time-out.

<90> Section 3.2.6.1: The Windows-based client will not disconnect the connection.

<91> Section 3.2.6.2: The Windows-based client will disconnect idle connections that have no open files and that have had no activity for 10 seconds.

<92> Section 3.3.1.2: Windows-based servers will limit the maximum range of sequence numbers. If a client has been granted 10 credits, the server will not allow the difference between the smallest available sequence number and the largest available sequence number to exceed 2*10 = 20. Therefore, if the client has sequence number 10 available and does not send it, the server will stop granting credits as the client nears sequence number 30, and eventually will grant no further credits until the client sends sequence number 10.

<93> Section 3.3.1.3: A Windows-based server will grant some portion of the client request based on available resources and the number of credits the client is currently taking advantage of. A Windows-based server grants credits based on usage but will attempt to enforce fairness if there are insufficient credits.

<94> Section 3.3.1.3: A Windows-based server does not currently scale credits based on quality of service features.

<95> Section 3.3.1.6: Windows-based servers allow the sharing of both printers and traditional file shares.

<96> Section 3.3.1.6: In Windows, this abstract state element contains the security descriptor for the share.

<97> Section 3.3.2.1: This timer has a default value of 35 seconds, but its value could be changed by system policy to any range between 5 seconds and infinite (4,294,967,295 seconds).

<98> Section 3.3.2.2: This timer has a default value of 16 minutes, but its value could be changed by system policy to any range between 5 seconds and infinite (4,294,967,295 seconds).

<99> Section 3.3.3: By default, Windows-based servers listen on both TCP-445 and NetBIOS. They can be configured to use only TCP-445 by disabling NetBIOS over TCP. Likewise, they can also pick up this setting via policy or DHCP configuration.

<100> Section 3.3.3: Windows-based SMB2 servers generate a new ServerGuid each time they are started.

<101> Section 3.3.4.1.2: Windows Vista and Windows Server 2008 do not grant credits on asynchronous responses. An interim response sent as asynchronous response for a SMB2 CHANGE_NOTIFY request will grant credits to keep the transaction from stalling in case the client is out of credits.

<102> Section 3.3.4.1.3: The Windows-based server compounds responses for any received compounded operations. Otherwise, it does not compound responses.

<103> Section 3.3.4.2: Windows-based servers send interim responses for the following operations if they cannot be completed immediately:

- SMB2 Create, when the server is in oplock break for the file under consideration
- SMB2 Change Notify
- Byte Range Lock
- Named Pipe Read on a blocking named pipe
- Named Pipe Write on a blocking named pipe
- FSCTL_PIPE_TRANSCEIVE

Windows servers allot a certain number of blocking operations with a statically configured blocking operation credit, which defaults to 50 but is configurable. If a client attempts to exceed this maximum number of blocking operations, the server will fail the request.

<104> Section 3.3.4.2: Windows servers always fail requests that go to async mode in a compound request with STATUS_INVALID_PARAMETER.

<105> Section 3.3.5.2.2: Windows-based servers will not fail the request with an error code but will disconnect clients who send requests with an invalid MessageId.

<106> Section 3.3.5.2.3: Windows-based servers will not disconnect the connection due to a mismatched signature.

<107> Section 3.3.5.2.3: Windows-based servers will not disconnect the connection due to an unsigned packet.

<108> Section 3.3.5.2.4: Windows-based servers will disconnect the connection when it processes packets that are smaller than the SMB2 header or packets that contain an invalid SMB2 command. For all other validations, it will not disconnect the connection but simply return the error.

<109> Section 3.3.5.2.5: Windows servers always fail requests that go to async mode in a compound request with STATUS_INVALID_PARAMETER except for the following two cases: when a create request in the compound request triggers an oplock break, or when the last request in the compound request wants to go async mode.

<110> Section 3.3.5.3: When a Windows–based client sends the deprecated "SMB2.001" dialect, a Windows Vista RTM–based server would acknowledge with a value of 6 in DialectRevision in SMB2 NEGOTIATE Response. This behavior is deprecated.

<111> Section 3.3.5.3: A Windows Vista RTM–based Server MUST set DialectRevision to 6.

<112> Section 3.3.5.3: Windows clients do not enforce the MaxTransactSize value.

<113> Section 3.3.5.4: Windows-based SMB2 server implementations allow multiple negotiate requests on the same connection.

<114> Section 3.3.5.4: A Windows Vista RTM–based Server checks to see if the dialect number zero was provided in the Dialect array of the SMB2_REQ_NEGOTIATE request. If it is not found, the server MUST fail the request with STATUS_NOT_SUPPORTED.

<115> Section 3.3.5.9.2: Windows will ignore security descriptors if the underlying file system does not support them.

<116> Section 3.3.5.9.3: Windows-based servers support this request.

<117> Section 3.3.5.11: Windows-based servers handle this request asynchronously.

<118> Section 3.3.5.12: Windows-based servers handle the following commands asynchronously: SMB2 Create (section 2.2.13) when this create would result in an oplock break, SMB2 IOCTL Request (section 2.2.31) for FSCTL_PIPE_TRANSCEIVE if it blocks for more than 1 millisecond, SMB2 Change Notify Request (section 2.2.35) if it blocks for more than 0.5 milliseconds, SMB2 Read request (section 2.2.19) for named pipes if it blocks for more than 0.5 milliseconds, SMB2 Write request (section 2.2.21) for named pipes if it blocks for more than 0.5 millseconds, and SMB2 lock request (section 2.2.26) if the SMB2_LOCKFLAG_FAIL_IMMEDIATELY flag is not set.

<119> Section 3.3.5.14.1: Windows-based servers ignore this value while processing Unlocks.

<120> Section 3.3.5.14.2: Windows-based servers check for SMB2_LOCKFLAG_FAIL_IMMEDIATELY only for the first element of the Locks array.

<121> Section 3.3.5.15.1: Windows clients set the InputOffset field to the offset, in bytes, from the beginning of the SMB2 header to the Buffer[] field of the response.

<122> Section 3.3.5.15.2: Windows clients set the InputOffset field to the offset, in bytes, from the beginning of the SMB2 header to the Buffer[] field of the response.

<123> Section 3.3.5.15.3: Windows clients set the InputOffset field to the offset, in bytes, from the beginning of the SMB2 header to the Buffer[] field of the response.

<124> Section 3.3.5.15.3: A Windows-based server currently returns the input buffer that is received in the request as part of the response.

<125> Section 3.3.5.15.4: Windows clients set the InputOffset field to the offset, in bytes, from the beginning of the SMB2 header to the Buffer[] field of the response.

<126> Section 3.3.5.15.5: Windows servers do not support any additional contexts.

<127> Section 3.3.5.15.5: Windows servers construct the 24-byte blob using the Open.Durable field and other pieces of information which include the process ID of the caller and a timestamp.

<128> Section 3.3.5.15.5: Windows clients set the InputOffset field to the offset, in bytes, from the beginning of the SMB2 header to the Buffer[] field of the response.

<129> Section 3.3.5.15.6: Windows servers will return STATUS_INVALID_VIEW_SIZE instead of STATUS_END_OF_FILE.

<130> Section 3.3.5.15.6: Windows clients set the InputOffset field to the offset, in bytes, from the beginning of the SMB2 header to the Buffer[] field of the response.

<131> Section 3.3.5.15.6.1: Windows clients set the InputOffset field to the offset, in bytes, from the beginning of the SMB2 header to the Buffer[] field of the response.

<132> Section 3.3.5.15.6.2: Windows clients set the InputOffset field to the offset, in bytes, from the beginning of the SMB2 header to the Buffer[] field of the response.

<133> Section 3.3.5.15.6.2: Windows Server 2003, Windows Vista, and Windows Server 2008 will accept a maximum of 256 chunks in a single operation, each chunk must be less than or equal to 1 megabyte, and the total data size must be less than or equal to 16 megabytes.

<134> Section 3.3.5.15.7: Windows-based SMB2 servers do not support IOCTL requests.

<135> Section 3.3.5.15.7: Windows clients set the InputOffset field to the offset, in bytes, from the beginning of the SMB2 header to the Buffer[] field of the response.

<136> Section 3.3.5.18: NTFS does not support resuming queries by index number that are provided by the client.

<137> Section 3.3.5.19: Windows-based servers handle the following commands asynchronously: SMB2 Create (section 2.2.13) when this create would result in an oplock break, SMB2 IOCTL Request (section 2.2.31) for FSCTL_PIPE_TRANSCEIVE if it blocks for more than 1 millisecond, SMB2 Change Notify Request (section 2.2.35) if it blocks for more than 0.5 milliseconds, SMB2 Read request (section 2.2.19) for named pipes if it blocks for more than 0.5 milliseconds, SMB2 Write request (section 2.2.21) for named pipes if it blocks for more than 0.5 millseconds, and SMB2 lock request (section 2.2.26) if the SMB2_LOCKFLAG_FAIL_IMMEDIATELY flag is not set.

<138> Section 3.3.5.20.4: Windows-based servers do support quotas, if configured.

<139> Section 3.3.5.21.3: If the underlying object store does not support object security based on Access Control Lists (as specified in [MS-DTYP] section **2.4.5**), it MUST return STATUS_SUCCESS.

<140> Section 3.3.5.21.4: Windows servers do support quotas, if configured.

7 Index

P

R

S

T

V

W