

[MS-SMB2]: Server Message Block (SMB) Version 2 Protocol Specification

Intellectual Property Rights Notice for Open Specifications Documentation

- **Technical Documentation.** Microsoft publishes Open Specifications documentation for protocols, file formats, languages, standards as well as overviews of the interaction among each of these technologies.
- **Copyrights.** This documentation is covered by Microsoft copyrights. Regardless of any other terms that are contained in the terms of use for the Microsoft website that hosts this documentation, you may make copies of it in order to develop implementations of the technologies described in the Open Specifications and may distribute portions of it in your implementations using these technologies or your documentation as necessary to properly document the implementation. You may also distribute in your implementation, with or without modification, any schema, IDL's, or code samples that are included in the documentation. This permission also applies to any documents that are referenced in the Open Specifications.
- **No Trade Secrets.** Microsoft does not claim any trade secret rights in this documentation.
- **Patents.** Microsoft has patents that may cover your implementations of the technologies described in the Open Specifications. Neither this notice nor Microsoft's delivery of the documentation grants any licenses under those or any other Microsoft patents. However, a given Open Specification may be covered by Microsoft [Open Specification Promise](#) or the [Community Promise](#). If you would prefer a written license, or if the technologies described in the Open Specifications are not covered by the Open Specifications Promise or Community Promise, as applicable, patent licenses are available by contacting iplg@microsoft.com.
- **Trademarks.** The names of companies and products contained in this documentation may be covered by trademarks or similar intellectual property rights. This notice does not grant any licenses under those rights.
- **Fictitious Names.** The example companies, organizations, products, domain names, e-mail addresses, logos, people, places, and events depicted in this documentation are fictitious. No association with any real company, organization, product, domain name, email address, logo, person, place, or event is intended or should be inferred.

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

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

Revision Summary

Date	Revision History	Revision Class	Comments
03/14/2007	1.0		Version 1.0 release
04/10/2007	1.1		Version 1.1 release
05/18/2007	1.2		Version 1.2 release
06/08/2007	2.0	Major	Updated and revised the technical content.
07/10/2007	2.1	Minor	Updated the technical content.
08/17/2007	3.0	Major	Updated and revised the technical content.
09/21/2007	4.0	Major	Updated and revised the technical content.
10/26/2007	5.0	Major	Updated and revised the technical content.
01/25/2008	5.0.1	Editorial	Revised and edited the technical content.
03/14/2008	6.0	Major	Updated and revised the technical content.
06/20/2008	7.0	Major	Updated and revised the technical content.
07/25/2008	8.0	Major	Updated and revised the technical content.
08/29/2008	9.0	Major	Updated and revised the technical content.
10/24/2008	10.0	Major	Updated and revised the technical content.
12/05/2008	11.0	Major	Updated and revised the technical content.
01/16/2009	12.0	Major	Updated and revised the technical content.
02/27/2009	13.0	Major	Updated and revised the technical content.
04/10/2009	14.0	Major	Updated and revised the technical content.
05/22/2009	15.0	Major	Updated and revised the technical content.
07/02/2009	16.0	Major	Updated and revised the technical content.
08/14/2009	17.0	Major	Updated and revised the technical content.
09/25/2009	18.0	Major	Updated and revised the technical content.
11/06/2009	19.0	Major	Updated and revised the technical content.
12/18/2009	20.0	Major	Updated and revised the technical content.
01/29/2010	21.0	Major	Updated and revised the technical content.
03/12/2010	22.0	Major	Updated and revised the technical content.
04/23/2010	23.0	Major	Updated and revised the technical content.

Date	Revision History	Revision Class	Comments
06/04/2010	24.0	Major	Updated and revised the technical content.
07/16/2010	25.0	Major	Significantly changed the technical content.
08/27/2010	26.0	Major	Significantly changed the technical content.
10/08/2010	27.0	Major	Significantly changed the technical content.
11/19/2010	28.0	Major	Significantly changed the technical content.
01/07/2011	29.0	Major	Significantly changed the technical content.
02/11/2011	30.0	Major	Significantly changed the technical content.
03/25/2011	31.0	Major	Significantly changed the technical content.
05/06/2011	32.0	Major	Significantly changed the technical content.
06/17/2011	33.0	Major	Significantly changed the technical content.

Contents

1	Introduction	12
1.1	Glossary	12
1.2	References.....	15
1.2.1	Normative References.....	15
1.2.2	Informative References	16
1.3	Overview	16
1.4	Relationship to Other Protocols.....	18
1.5	Prerequisites/Preconditions	19
1.6	Applicability Statement.....	20
1.7	Versioning and Capability Negotiation.....	20
1.8	Vendor-Extensible Fields.....	21
1.9	Standards Assignments	22
2	Messages.....	23
2.1	Transport.....	23
2.2	Message Syntax	23
2.2.1	SMB2 Packet Header.....	24
2.2.1.1	SMB2 Packet Header - ASYNC	24
2.2.1.2	SMB2 Packet Header - SYNC	27
2.2.2	SMB2 ERROR Response Packet.....	31
2.2.2.1	Symbolic Link Error Response	31
2.2.2.1.1	Handling the Symbolic Link Error Response	33
2.2.3	SMB2 NEGOTIATE Request	35
2.2.4	SMB2 NEGOTIATE Response	36
2.2.5	SMB2 SESSION_SETUP Request.....	39
2.2.6	SMB2 SESSION_SETUP Response.....	40
2.2.7	SMB2 LOGOFF Request	41
2.2.8	SMB2 LOGOFF Response	42
2.2.9	SMB2 TREE_CONNECT Request	42
2.2.10	SMB2 TREE_CONNECT Response	43
2.2.11	SMB2 TREE_DISCONNECT Request	45
2.2.12	SMB2 TREE_DISCONNECT Response	45
2.2.13	SMB2 CREATE Request.....	45
2.2.13.1	SMB2 Access Mask Encoding	51
2.2.13.1.1	File_Pipe_Printer_Access_Mask	51
2.2.13.1.2	Directory_Access_Mask	52
2.2.13.2	SMB2_CREATE_CONTEXT Request Values	54
2.2.13.2.1	SMB2_CREATE_EA_BUFFER	56
2.2.13.2.2	SMB2_CREATE_SD_BUFFER.....	56
2.2.13.2.3	SMB2_CREATE_DURABLE_HANDLE_REQUEST.....	56
2.2.13.2.4	SMB2_CREATE_DURABLE_HANDLE_RECONNECT.....	56
2.2.13.2.5	SMB2_CREATE_QUERY_MAXIMAL_ACCESS_REQUEST.....	57
2.2.13.2.6	SMB2_CREATE_ALLOCATION_SIZE.....	57
2.2.13.2.7	SMB2_CREATE_TIMEWARP_TOKEN.....	58
2.2.13.2.8	SMB2_CREATE_REQUEST_LEASE	58
2.2.13.2.9	SMB2_CREATE_QUERY_ON_DISK_ID.....	59
2.2.14	SMB2 CREATE Response.....	59
2.2.14.1	SMB2_FILEID	62
2.2.14.2	SMB2_CREATE_CONTEXT Response Values	62
2.2.14.2.1	SMB2_CREATE_EA_BUFFER	62

2.2.14.2.2	SMB2_CREATE_SD_BUFFER.....	62
2.2.14.2.3	SMB2_CREATE_DURABLE_HANDLE_RESPONSE.....	63
2.2.14.2.4	SMB2_CREATE_DURABLE_HANDLE_RECONNECT.....	63
2.2.14.2.5	SMB2_CREATE_QUERY_MAXIMAL_ACCESS_RESPONSE.....	63
2.2.14.2.6	SMB2_CREATE_ALLOCATION_SIZE.....	63
2.2.14.2.7	SMB2_CREATE_TIMEWARP_TOKEN.....	64
2.2.14.2.8	SMB2_CREATE_QUERY_ON_DISK_ID.....	64
2.2.14.2.9	SMB2_CREATE_RESPONSE_LEASE.....	64
2.2.15	SMB2 CLOSE Request.....	65
2.2.16	SMB2 CLOSE Response.....	66
2.2.17	SMB2 FLUSH Request.....	68
2.2.18	SMB2 FLUSH Response.....	69
2.2.19	SMB2 READ Request.....	69
2.2.20	SMB2 READ Response.....	70
2.2.21	SMB2 WRITE Request.....	71
2.2.22	SMB2 WRITE Response.....	73
2.2.23	SMB2 OPLOCK_BREAK Notification.....	74
2.2.23.1	Oplock Break Notification.....	74
2.2.23.2	Lease Break Notification.....	75
2.2.24	SMB2 OPLOCK_BREAK Acknowledgment.....	76
2.2.24.1	Oplock Break Acknowledgment.....	76
2.2.24.2	Lease Break Acknowledgment.....	77
2.2.25	SMB2 OPLOCK_BREAK Response.....	78
2.2.25.1	Oplock Break Response.....	78
2.2.25.2	Lease Break Response.....	79
2.2.26	SMB2 LOCK Request.....	80
2.2.26.1	SMB2_LOCK_ELEMENT Structure.....	81
2.2.27	SMB2 LOCK Response.....	82
2.2.28	SMB2 ECHO Request.....	83
2.2.29	SMB2 ECHO Response.....	83
2.2.30	SMB2 CANCEL Request.....	83
2.2.31	SMB2 IOCTL Request.....	84
2.2.31.1	SRV_COPYCHUNK_COPY.....	86
2.2.31.1.1	SRV_COPYCHUNK.....	87
2.2.31.2	SRV_READ_HASH Request.....	87
2.2.31.2.1	HASH_HEADER.....	88
2.2.31.3	NETWORK_RESILIENCY_REQUEST Request.....	90
2.2.32	SMB2 IOCTL Response.....	90
2.2.32.1	SRV_COPYCHUNK_RESPONSE.....	92
2.2.32.2	SRV_SNAPSHOT_ARRAY.....	92
2.2.32.3	SRV_REQUEST_RESUME_KEY Response.....	93
2.2.32.4	SRV_READ_HASH Response.....	94
2.2.33	SMB2 QUERY_DIRECTORY Request.....	94
2.2.34	SMB2 QUERY_DIRECTORY Response.....	96
2.2.35	SMB2 CHANGE_NOTIFY Request.....	97
2.2.36	SMB2 CHANGE_NOTIFY Response.....	99
2.2.36.1	FILE_NOTIFY_INFORMATION.....	99
2.2.37	SMB2 QUERY_INFO Request.....	100
2.2.37.1	SMB2_QUERY_QUOTA_INFO.....	104
2.2.38	SMB2 QUERY_INFO Response.....	105
2.2.39	SMB2 SET_INFO Request.....	106
2.2.40	SMB2 SET_INFO Response.....	109

3 Protocol Details	110
3.1 Common Details	110
3.1.1 Abstract Data Model	110
3.1.1.1 Global	110
3.1.2 Timers	110
3.1.3 Initialization	110
3.1.4 Higher-Layer Triggered Events	110
3.1.4.1 Signing An Outgoing Message	110
3.1.5 Processing Events and Sequencing Rules	111
3.1.5.1 Verifying an Incoming Message	111
3.1.6 Timer Events	111
3.1.7 Other Local Events	111
3.2 Client Details	111
3.2.1 Abstract Data Model	111
3.2.1.1 Global	111
3.2.1.2 Per SMB2 Transport Connection	112
3.2.1.3 Per Session	113
3.2.1.4 Per Tree Connect	113
3.2.1.5 Per Open File	113
3.2.1.6 Per Application Open of a File	114
3.2.1.7 Per Pending Request	114
3.2.2 Timers	115
3.2.2.1 Request Expiration Timer	115
3.2.2.2 Idle Connection Timer	115
3.2.3 Initialization	115
3.2.4 Higher-Layer Triggered Events	115
3.2.4.1 Sending Any Outgoing Message	115
3.2.4.1.1 Signing the Message	116
3.2.4.1.2 Requesting Credits from the Server	116
3.2.4.1.3 Associating the Message with a MessageId	116
3.2.4.1.4 Sending Compounded Requests	117
3.2.4.1.5 Sending Multi-Credit Requests	117
3.2.4.1.6 Algorithm for Handling Available Message Sequence Numbers by the Client	118
3.2.4.2 Application Requests a Connection to a Share	118
3.2.4.2.1 Connecting to the Target Server	120
3.2.4.2.2 Negotiating the Protocol	121
3.2.4.2.2.1 Multi-Protocol Negotiate	121
3.2.4.2.2.2 SMB2-Only Negotiate	121
3.2.4.2.3 Authenticating the User	122
3.2.4.2.3.1 Application Requests Reauthenticating a User	123
3.2.4.2.4 Connecting to the Share	124
3.2.4.3 Application Requests Opening a File	124
3.2.4.3.1 Application Requests Opening a Named Pipe	126
3.2.4.3.2 Application Requests Sending a File to Print	126
3.2.4.3.3 Application Requests Creating a File with Extended Attributes	126
3.2.4.3.4 Application Requests Creating a File with a Security Descriptor	127
3.2.4.3.5 Application Requests Creating a File Opened for Durable Operation	127
3.2.4.3.6 Application Requests Opening a Previous Version of a File	127
3.2.4.3.7 Application Requests Creating a File with a Specific Allocation Size	127
3.2.4.3.8 Requesting a Lease on a File	127
3.2.4.3.9 Application Requests Maximal Access Information of a File	127
3.2.4.3.10 Application Requests Identifier of a File	128

3.2.4.4	Re-establishing a Durable Open.....	128
3.2.4.5	Application Requests Closing a File or Named Pipe	129
3.2.4.6	Application Requests Reading from a File or Named Pipe	129
3.2.4.7	Application Requests Writing to a File or Named Pipe	130
3.2.4.8	Application Requests Querying File Attributes.....	131
3.2.4.9	Application Requests Applying File Attributes.....	133
3.2.4.10	Application Requests Querying File System Attributes.....	134
3.2.4.11	Application Requests Applying File System Attributes	135
3.2.4.12	Application Requests Querying File Security	136
3.2.4.13	Application Requests Applying File Security	137
3.2.4.14	Application Requests Querying Quota Information.....	137
3.2.4.15	Application Requests Applying Quota Information	139
3.2.4.16	Application Requests Flushing Cached Data	140
3.2.4.17	Application Requests Enumerating a Directory	141
3.2.4.17.1	Application Requests Continuing a Directory Enumeration.....	142
3.2.4.18	Application Requests Change Notifications for a Directory	142
3.2.4.19	Application Requests Locking of an Array of Byte Ranges	143
3.2.4.20	Application Requests an IO Control Code Operation.....	144
3.2.4.20.1	Application Requests Enumeration of Previous Versions.....	144
3.2.4.20.2	Application Requests a Server-Side Data Copy	145
3.2.4.20.2.1	Application Requests a Source File Key	146
3.2.4.20.2.2	Application Requests a Server Side Data Copy	147
3.2.4.20.3	Application Requests DFS Referral Information.....	148
3.2.4.20.4	Application Requests a Pipe Transaction	150
3.2.4.20.5	Application Requests a Peek at Pipe Data	151
3.2.4.20.6	Application Requests a Pass-Through Operation	152
3.2.4.20.7	Application Requests Content Information for a File	153
3.2.4.20.8	Application Requests Resiliency on an Open File	154
3.2.4.20.9	Application Requests Waiting for a Connection to a Pipe	155
3.2.4.21	Application Requests Unlocking of an Array of Byte Ranges	156
3.2.4.22	Application Requests Closing a Share Connection	158
3.2.4.23	Application Requests Terminating an Authenticated Context.....	158
3.2.4.24	Application Requests Canceling an Operation.....	158
3.2.4.25	Application Requests the Session Key for an Authenticated Context	159
3.2.4.26	Application Requests Number of Opens on a Tree Connect	159
3.2.5	Processing Events and Sequencing Rules.....	159
3.2.5.1	Receiving Any Message	159
3.2.5.1.1	Finding the Application Request for This Response.....	159
3.2.5.1.2	Verifying the Signature	160
3.2.5.1.3	Granting Message Credits.....	160
3.2.5.1.4	Handling Asynchronous Responses	160
3.2.5.1.5	Handling Session Expiration.....	160
3.2.5.1.6	Handling Incorrectly Formatted Responses	161
3.2.5.1.7	Processing the Response	161
3.2.5.1.8	Handling Compounded Responses	161
3.2.5.2	Receiving an SMB2 NEGOTIATE Response.....	161
3.2.5.3	Receiving an SMB2 SESSION_SETUP Response	162
3.2.5.3.1	Handling a New Authentication	162
3.2.5.3.2	Handling a Reauthentication	163
3.2.5.4	Receiving an SMB2 LOGOFF Response.....	164
3.2.5.5	Receiving an SMB2 TREE_CONNECT Response	164
3.2.5.6	Receiving an SMB2 TREE_DISCONNECT Response	165
3.2.5.7	Receiving an SMB2 CREATE Response for a New Create Operation.....	165

3.2.5.7.1	SMB2_CREATE_DURABLE_HANDLE_RESPONSE Create Context.....	166
3.2.5.7.2	SMB2_CREATE_QUERY_MAXIMAL_ACCESS_RESPONSE Create Context	166
3.2.5.7.3	SMB2_CREATE_QUERY_ON_DISK_ID Create Context	167
3.2.5.7.4	SMB2_CREATE_RESPONSE_LEASE Create Context	167
3.2.5.8	Receiving an SMB2 CREATE Response for an Open Reestablishment	167
3.2.5.9	Receiving an SMB2 CLOSE Response.....	168
3.2.5.10	Receiving an SMB2 FLUSH Response	168
3.2.5.11	Receiving an SMB2 READ Response.....	168
3.2.5.12	Receiving an SMB2 WRITE Response	168
3.2.5.13	Receiving an SMB2 LOCK Response.....	168
3.2.5.14	Receiving an SMB2 IOCTL Response.....	169
3.2.5.14.1	Handling an Enumeration of Previous Versions Response.....	169
3.2.5.14.2	Handling a Server-Side Data Copy Source File Key Response	169
3.2.5.14.3	Handling a Server-Side Data Copy Response	169
3.2.5.14.4	Handling a DFS Referral Information Response.....	169
3.2.5.14.5	Handling a Pipe Transaction Response.....	170
3.2.5.14.6	Handling a Peek at Pipe Data Response	170
3.2.5.14.7	Handling a Content Information Retrieval Response.....	170
3.2.5.14.8	Handling a Pass-Through Operation Response.....	170
3.2.5.14.9	Handling a Resiliency Response.....	171
3.2.5.14.10	Handling a Pipe Wait Response.....	171
3.2.5.15	Receiving an SMB2 QUERY_DIRECTORY Response	171
3.2.5.16	Receiving an SMB2 CHANGE_NOTIFY Response	171
3.2.5.17	Receiving an SMB2 QUERY_INFO Response	171
3.2.5.18	Receiving an SMB2 SET_INFO Response	172
3.2.5.19	Receiving an SMB2 OPLOCK_BREAK Notification	172
3.2.5.19.1	Receiving an Oplock Break Notification.....	172
3.2.5.19.2	Receiving a Lease Break Notification	173
3.2.5.19.3	Receiving an Oplock Break Acknowledgment Response	174
3.2.5.19.4	Receiving a Lease Break Acknowledgment Response.....	174
3.2.6	Timer Events	174
3.2.6.1	Request Expiration Timer Event.....	174
3.2.6.2	Idle Connection Timer Event	174
3.2.7	Other Local Events	175
3.2.7.1	Handling a Network Disconnect	175
3.3	Server Details	175
3.3.1	Abstract Data Model	175
3.3.1.1	Algorithm for Handling Available Message Sequence Numbers by the Server...	176
3.3.1.2	Algorithm for the Granting of Credits.....	176
3.3.1.3	Algorithm for Change Notifications in an Object Store	177
3.3.1.4	Algorithm for Leasing in an Object Store.....	177
3.3.1.5	Global Structures.....	178
3.3.1.6	Per Share Structures.....	179
3.3.1.7	Per Transport Connection Structures	181
3.3.1.8	Per Session Structures	181
3.3.1.9	Per Tree Connect Structures	182
3.3.1.10	Per Open Structures.....	182
3.3.1.11	Per Lease Table Structures.....	184
3.3.1.12	Per Lease Structures	184
3.3.1.13	Per Request Structures	185
3.3.2	Timers	185
3.3.2.1	Oplock Break Acknowledgment Timer.....	185
3.3.2.2	Durable Open Scavenger Timer	186

3.3.2.3	Session Expiration Timer	186
3.3.2.4	Resilient Open Scavenger Timer	186
3.3.3	Initialization	186
3.3.4	Higher-Layer Triggered Events	187
3.3.4.1	Sending Any Outgoing Message	187
3.3.4.1.1	Signing the Message	187
3.3.4.1.2	Granting Credits to the Client	187
3.3.4.1.3	Sending Compounded Responses	188
3.3.4.2	Sending an Interim Response for an Asynchronous Operation	188
3.3.4.3	Sending a Success Response	189
3.3.4.4	Sending an Error Response	190
3.3.4.5	Server Application Requests Session Key of the Client	190
3.3.4.6	Object Store Indicates an Oplock Break	191
3.3.4.7	Object Store Indicates a Lease Break	191
3.3.4.8	DFS Server Notifies SMB2 Server That DFS Is Active	192
3.3.4.9	DFS Server Notifies SMB2 Server That a Share Is a DFS Share	192
3.3.4.10	DFS Server Notifies SMB2 Server That a Share Is Not a DFS Share	192
3.3.4.11	Server Application Requests Security Context of the Client	192
3.3.4.12	Server Application Requests Closing a Session	193
3.3.4.13	Server Application Registers a Share	193
3.3.4.14	Server Application Updates a Share	194
3.3.4.15	Server Application Deregisters a Share	195
3.3.4.16	Server Application Requests Querying a Share	195
3.3.4.17	Server Application Requests Closing an Open	196
3.3.4.18	Server Application Queries a Session	197
3.3.4.19	Server Application Queries a TreeConnect	198
3.3.4.20	Server Application Queries an Open	198
3.3.4.21	Server Application Requests Transport Binding Change	198
3.3.4.22	Server Application Enables the SMB2 Server	199
3.3.4.23	Server Application Disables the SMB2 Server	199
3.3.4.24	Server Application Requests Server Statistics	199
3.3.5	Processing Events and Sequencing Rules	200
3.3.5.1	Accepting an Incoming Connection	200
3.3.5.2	Receiving Any Message	201
3.3.5.2.1	Verifying the Connection State	201
3.3.5.2.2	Verifying the Sequence Number	201
3.3.5.2.3	Verifying the Signature	202
3.3.5.2.4	Verifying the Credit Charge and the Payload Size	202
3.3.5.2.5	Handling Incorrectly Formatted Requests	203
3.3.5.2.6	Handling Compounded Requests	203
3.3.5.2.6.1	Handling Compounded Unrelated Requests	203
3.3.5.2.6.2	Handling Compounded Related Requests	203
3.3.5.2.7	Updating Idle Time	204
3.3.5.3	Receiving an SMB_COM_NEGOTIATE	204
3.3.5.3.1	SMB 2.1 Support	204
3.3.5.3.2	SMB 2.002 Support	205
3.3.5.4	Receiving an SMB2 NEGOTIATE Request	206
3.3.5.5	Receiving an SMB2 SESSION_SETUP Request	208
3.3.5.5.1	Authenticating a New Session	208
3.3.5.5.2	Reauthenticating an Existing Session	209
3.3.5.5.3	Handling GSS-API Authentication	209
3.3.5.6	Receiving an SMB2 LOGOFF Request	213
3.3.5.7	Receiving an SMB2 TREE_CONNECT Request	213

3.3.5.8	Receiving an SMB2 TREE_DISCONNECT Request	215
3.3.5.9	Receiving an SMB2 CREATE Request	216
3.3.5.9.1	Handling the SMB2_CREATE_EA_BUFFER Create Context	221
3.3.5.9.2	Handling the SMB2_CREATE_SD_BUFFER Create Context	221
3.3.5.9.3	Handling the SMB2_CREATE_ALLOCATION_SIZE Create Context	222
3.3.5.9.4	Handling the SMB2_CREATE_TIMEWARP_TOKEN Create Context	222
3.3.5.9.5	Handling the SMB2_CREATE_QUERY_MAXIMAL_ACCESS_REQUEST Create Context	222
3.3.5.9.6	Handling the SMB2_CREATE_DURABLE_HANDLE_REQUEST Create Context	223
3.3.5.9.7	Handling the SMB2_CREATE_DURABLE_HANDLE_RECONNECT Create Context	223
3.3.5.9.8	Handling the SMB2_CREATE_REQUEST_LEASE Create Context	224
3.3.5.9.9	Handling the SMB2_CREATE_QUERY_ON_DISK_ID Create Context	226
3.3.5.10	Receiving an SMB2 CLOSE Request	226
3.3.5.11	Receiving an SMB2 FLUSH Request	227
3.3.5.12	Receiving an SMB2 READ Request	228
3.3.5.13	Receiving an SMB2 WRITE Request	230
3.3.5.14	Receiving an SMB2 LOCK Request	231
3.3.5.14.1	Processing Unlocks	232
3.3.5.14.2	Processing Locks	233
3.3.5.15	Receiving an SMB2 IOCTL Request	234
3.3.5.15.1	Handling an Enumeration of Previous Versions Request	236
3.3.5.15.2	Handling a DFS Referral Information Request	237
3.3.5.15.3	Handling a Pipe Transaction Request	238
3.3.5.15.4	Handling a Peek at Pipe Data Request	238
3.3.5.15.5	Handling a Source File Key Request	239
3.3.5.15.6	Handling a Server-Side Data Copy Request	240
3.3.5.15.6.1	Sending a Copy Failure Server-Side Copy Response	241
3.3.5.15.6.2	Sending an Invalid Parameter Server-Side Copy Response	241
3.3.5.15.7	Handling a Content Information Retrieval Request	242
3.3.5.15.8	Handling a Pass-Through Operation Request	243
3.3.5.15.9	Handling a Resiliency Request	244
3.3.5.15.10	Handling a Pipe Wait Request	245
3.3.5.16	Receiving an SMB2 CANCEL Request	246
3.3.5.17	Receiving an SMB2 ECHO Request	246
3.3.5.18	Receiving an SMB2 QUERY_DIRECTORY Request	246
3.3.5.19	Receiving an SMB2 CHANGE_NOTIFY Request	249
3.3.5.20	Receiving an SMB2 QUERY_INFO Request	251
3.3.5.20.1	Handling SMB2_0_INFO_FILE	252
3.3.5.20.2	Handling SMB2_0_INFO_FILESYSTEM	253
3.3.5.20.3	Handling SMB2_0_INFO_SECURITY	254
3.3.5.20.4	Handling SMB2_0_INFO_QUOTA	255
3.3.5.21	Receiving an SMB2 SET_INFO Request	256
3.3.5.21.1	Handling SMB2_0_INFO_FILE	257
3.3.5.21.2	Handling SMB2_0_INFO_FILESYSTEM	258
3.3.5.21.3	Handling SMB2_0_INFO_SECURITY	258
3.3.5.21.4	Handling SMB2_0_INFO_QUOTA	259
3.3.5.22	Receiving an SMB2 OPLOCK_BREAK Acknowledgment	259
3.3.5.22.1	Processing an Oplock Acknowledgment	260
3.3.5.22.2	Processing a Lease Acknowledgment	261
3.3.6	Timer Events	262
3.3.6.1	Oplock Break Acknowledgment Timer Event	262

3.3.6.2	Durable Open Scavenger Timer Event	262
3.3.6.3	Session Expiration Timer Event	262
3.3.6.4	Resilient Open Scavenger Timer Event	262
3.3.7	Other Local Events	263
3.3.7.1	Handling Loss of a Connection	263
4	Protocol Examples.....	265
4.1	Connecting to a Share by Using a Multi-Protocol Negotiate.....	265
4.2	Negotiating SMB 2.10 dialect by using Multi-Protocol Negotiate	271
4.3	Connecting to a Share by Using an SMB2 Negotiate	276
4.4	Executing an Operation on a Named Pipe.....	282
4.5	Reading from a Remote File	291
4.6	Writing to a Remote File	297
4.7	Disconnecting a Share and Logging Off.....	307
5	Security.....	310
5.1	Security Considerations for Implementers.....	310
5.2	Index of Security Parameters	310
6	Appendix A: Product Behavior.....	311
7	Change Tracking.....	334
8	Index	336

1 Introduction

This specification describes the Server Message Block (SMB) Version 2 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 security concepts described in [\[MS-WSO\]](#) section 3.1.2.

1.1 Glossary

The following terms are defined in [\[MS-GLOS\]](#):

@GMT token
discretionary access control list (DACL)
Distributed File System (DFS)
Distributed File System (DFS) link
file system
file system control (FSCTL)
fully qualified domain name (FQDN)(1)
globally unique identifier (GUID)
guest account
handle
main stream
named pipe
named stream
NetBIOS
NT file system (NTFS)
print job
reparse point
security context
security descriptor
security identifier (SID)
security principal
snapshot
symbolic link
system access control list (SACL)
Transmission Control Protocol (TCP)
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.

Branch Cache: Branch Cache is intended to reduce bandwidth consumption on branch-office wide area network (WAN) links. Branch Cache clients retrieve content from distributed caches within a branch instead of remote servers. Distributed caches in the branch can either be on peer clients within the branch or be on dedicated caching servers. Branch Cache details are discussed in [\[MS-PCCRR\]](#).

Compounded Requests and Responses: A method of combining multiple SMB 2 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 Protocol client and an SMB 2 Protocol server.

Content: A file that is accessed by an application. Examples of content include Web pages and documents stored on either Web Servers or SMB file servers.

Content Information: An opaque blob of data containing a set of hashes for a specific file that can be used by the application to retrieve the contents of the file using the branch cache. The details of Content Information are discussed in [\[MS-PCCRC\]](#).

Content Information File: A file that stores **Content Information** along with a HASH_HEADER (see section [2.2.31.2.1](#)).

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 Protocol client by an SMB 2 Protocol server that limits the number of outstanding requests that a client can send to a server.

Durable Open: An **open** to a file that allows the client to attempt to preserve and reestablish the **open** after a network disconnect. It cannot be permissible to a directory, named pipe, or printer.

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.

Lease: 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 confirms that no other client is accessing the data. Likewise, the client may buffer read-ahead data from the remote file if the client confirms that no other client is writing data to the remote file.

There are three types of leases:

- A read-caching lease allows a client to cache reads and can be granted to multiple clients.
- A write-caching lease allows a client to cache writes and byte range locks and can only be granted to a single client.
- A handle-caching lease allows a client to cache open handles and can be granted to multiple clients.

A lease can be a combination of one or more of the lease types listed above. When a client opens a file, it requests that the server grant it a lease on the file. The response from the server indicates the lease that is granted to the client. The client uses the granted lease to adjust its buffering policy.

A lease can span multiple opens as well as multiple connections from the same client.

Lease Break: An unsolicited request that is sent by an SMB 2 Protocol server to an SMB 2 Protocol client to inform the client to change the lease state for a file.

Local object store: A system that provides the ability to create, query, modify, or apply policy to a local resource on behalf of a remote client. The object store is backed by either a **file system**, a **named pipe**, or a **print job** that is accessed as a file.

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 confirms that no other process is accessing the data. Likewise, the client may buffer read-ahead data from the remote file if the client confirms 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 Protocol server to an SMB 2 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 Protocol **connection**. For a description of how **sequence numbers** are allocated, see sections [3.2.4.1.6](#) and [3.3.1.1](#).

Session: An **authenticated context** that is established between an SMB 2 Protocol client and an SMB 2 Protocol server over an SMB 2 Protocol **connection** for a specific security principal. There could be multiple active **sessions** over a single SMB 2 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 Protocol server for access by SMB 2 Protocol clients over the network. The SMB 2 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 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 Protocol client to a specific **share** on an SMB 2 Protocol server over an SMB 2 Protocol **connection**. There could be

multiple **tree connects** over a single SMB 2 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

References to Microsoft Open Specification documents do not include a publishing year because links are to the latest version of the documents, which are updated frequently. References to other documents include a publishing year when one is available.

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.

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

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

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

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

[MS-FSCC] Microsoft Corporation, "[File System Control Codes](#)".

[MS-KILE] Microsoft Corporation, "[Kerberos Protocol Extensions](#)".

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

[MS-PCCRC] Microsoft Corporation, "[Peer Content Caching and Retrieval: Content Identification](#)".

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

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

[MS-SPNG] Microsoft Corporation, "[Simple and Protected GSS-API Negotiation Mechanism \(SPNEGO\) Extension](#)".

[MS-SRVS] Microsoft Corporation, "[Server Service Remote Protocol Specification](#)".

[RFC1002] Network Working Group, "Protocol Standard for a NetBIOS Service on a TCP/UDP Transport: Detailed Specifications", STD 19, 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.rfc-editor.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

[FSBO] Microsoft Corporation, "File System Behavior in the Microsoft Windows Environment", June 2008, <http://download.microsoft.com/download/4/3/8/43889780-8d45-4b2e-9d3a-c696a890309f/File%20System%20Behavior%20Overview.pdf>

[MS-AUTHSO] Microsoft Corporation, "[Windows Authentication Services System Overview](#)".

[MS-FSA] Microsoft Corporation, "[File System Algorithms](#)".

[MS-GLOS] Microsoft Corporation, "[Windows Protocols Master Glossary](#)".

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

[MS-PCCRR] Microsoft Corporation, "[Peer Content Caching and Retrieval: Retrieval Protocol Specification](#)".

[MS-WSO] Microsoft Corporation, "[Windows System Overview](#)".

[MSDFS] Microsoft Corporation, "How DFS Works", March 2003, <http://technet.microsoft.com/en-us/library/cc782417%28WS.10%29.aspx>

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

[MSDN-IoCtlCodes] Microsoft Corporation, "Defining I/O Control Codes", <http://msdn.microsoft.com/en-us/library/ms795909.aspx>

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

1.3 Overview

The Server Message Block (SMB) Version 2 Protocol is an extension of the original [Server Message Block \(SMB\) Protocol](#) (as specified in [MS-SMB] and [\[MS-CIFS\]](#)). 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 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 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 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 underlying object store on the server machine.
- Validating the integrity of requests and responses.
- Support for share scoping and server aliases to allow a single server to appear as multiple distinct servers, as described in [\[MS-SRVS\]](#) section 1.3.

The SMB 2 Protocol provides several enhancements in addition to the preceding 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.
- Supporting symbolic links.
- Using a stronger algorithm to validate the integrity of requests and responses.

The SMB 2.1 dialect introduces the following enhancements:

- Allowing a client to indicate support for multiple SMB 2 dialects in a multi-protocol negotiate request.
- Allowing a client to obtain and preserve client caching state across multiple opens from the same client.
- Allowing a client to mark individual write operations on unbuffered handles to be treated as write-through.
- Allowing a client to retrieve hashes of a file for use in branch cache retrieval, as specified in [\[MS-PCCRC\]](#).

1.4 Relationship to Other Protocols

The Server Message Block (SMB) Version 2 Protocol may be negotiated by using an SMB negotiate, as specified in [\[MS-SMB\]](#) section 1.7. After a dialect of the SMB 2 Protocol is selected during negotiation, all messages that are sent on the connection (including the negotiate response) will be SMB 2 Protocol messages, as specified in this document, and no further SMB traffic will be exchanged on the connection.

For authentication, the SMB 2 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 Protocol uses either TCP or NetBIOS over TCP as underlying transports.

Machines using the SMB 2 Protocol can use 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.

DFS clients communicate with DFS servers via referral requests/responses conveyed in SMB2 **IOCTL** messages, analogous to a file system client performing control operations on a remote object store via requests/responses conveyed in SMB2 IOCTL messages. The communication between the SMB2 server and the DFS server (or SMB2 server and object store), for the purpose of performing the specified IOCTL operations, is local to the server machine, and takes place via implementation-dependent means.

The Remote Procedure Call Protocol Extensions, as specified in [\[MS-RPCE\]](#), define an RPC over SMB Protocol or SMB 2 Protocol sequence that can use SMB 2 Protocol named pipes as its underlying transport. The selection of protocol is based on client behavior during negotiation, as specified in section [1.7](#).

Peer Content Caching and Retrieval framework, or **Branch Cache**, is designed to reduce bandwidth consumption on branch-office wide area network (WAN) links by having clients request **Content** from distributed caches. Content is uniquely identified by **Content Information** retrieved from the server through SMB 2 IOCTL messages, as specified in sections [3.2.4.20.7](#) and [3.3.5.15.7](#). This capability is only supported for the SMB 2.1 dialect.

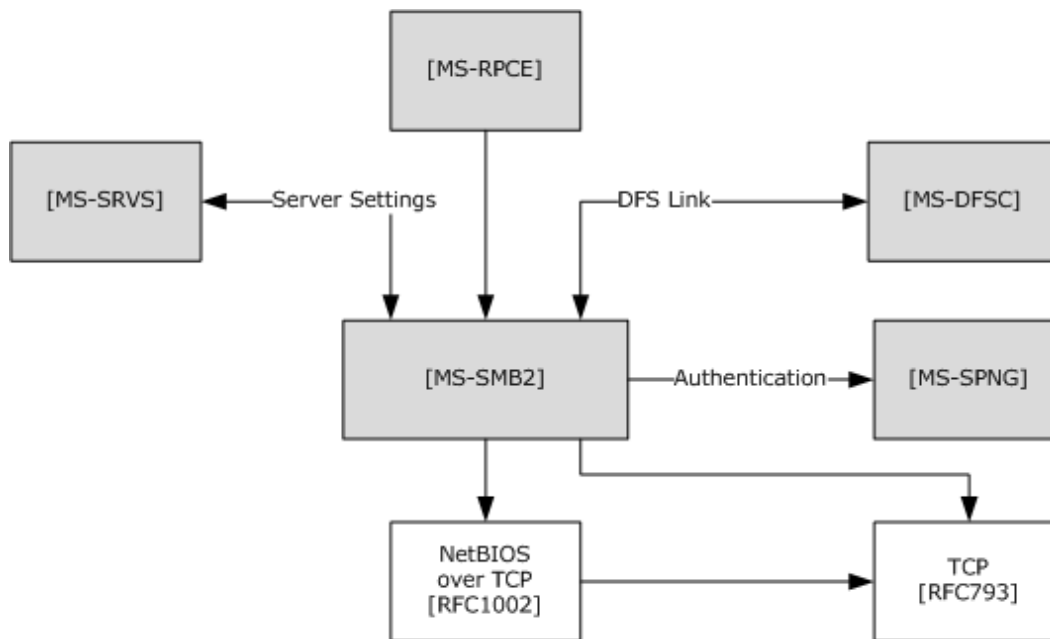


Figure 1: Relationship to other protocols

The diagram shows the following:

- [MS-RPCE] uses [MS-SMB2] named pipes as its underlying transport.
- [MS-DFSC] uses [MS-SMB2] as its transport layer.
- [\[MS-SRVS\]](#) calls [MS-SMB2] for file server management.
- [MS-SMB2] calls [MS-SPNG] for authenticating the user.
- [MS-SMB2] calls [MS-DFSC] to resolve names from a namespace.
- [MS-SMB2] calls [MS-SRVS] for server management and for synchronizing information on shares, sessions, treeconnects, and file opens. The synchronization mechanism is dependent on the SMB2 server and the server service starting up and terminating at the same time.
- [MS-SMB2] uses either TCP or NetBIOS over TCP as underlying transports.

1.5 Prerequisites/Preconditions

The Server Message Block (SMB) Version 2 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 Protocol [≤1>](#) is applicable for all scenarios that involve transferring files between client and server. The SMB 2 Protocol is also applicable for inter-process communication between client and server using named pipes.

The SMB 2 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 HMAC-SHA256 hash algorithm. The HMAC-SHA256 hash is specified in [\[FIPS180-2\]](#) and [\[RFC2104\]](#).
- 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 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 Protocol closely match the [SMB Protocol](#), as specified in [MS-SMB], the wire format for SMB 2 Protocol packets is different from that of the SMB Protocol. For maintaining interoperability between clients and servers in a mixed SMB 2/SMB Protocol environment, the SMB 2 Protocol can be negotiated in one of two ways:
 - By using an SMB negotiate message (as specified in [MS-SMB] sections [2.2.4.5.1](#) and [3.2.4.2.2](#)) with one of the new dialect strings: "SMB 2.002" or "SMB 2.???".
 - 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 Protocol-capable server that it requests to use SMB 2, 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 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 Protocol.

Servers capable of only the SMB 2 Protocol would reject communication with traditional SMB Protocol clients that do not offer "SMB 2.002" or "SMB 2.???" as a negotiate dialect, and accept communication only from SMB 2 Protocol clients.

There are currently two dialects of the SMB 2 Protocol. Negotiating the SMB 2.1 dialect implies support for the requests and responses as specified in this document, unless specifically noted otherwise. Negotiating the SMB 2.002 dialect implies support for the requests and responses as specified in this document, except those explicitly marked for the SMB 2.1 dialect. Certain functionalities are negotiated through the exchange of capabilities as specified in sections [2.2.4](#) and [2.2.5](#).

The following state diagram illustrates dialect negotiation in the SMB 2.1 server. In this diagram, state transitions occur as the SMB_COM_NEGOTIATE, SMB2 NEGOTIATE, and other requests are received from the client. The server uses a per-connection variable, **Connection.NegotiateDialect**, to represent the current state of dialect negotiation between client and server on each transport connection.

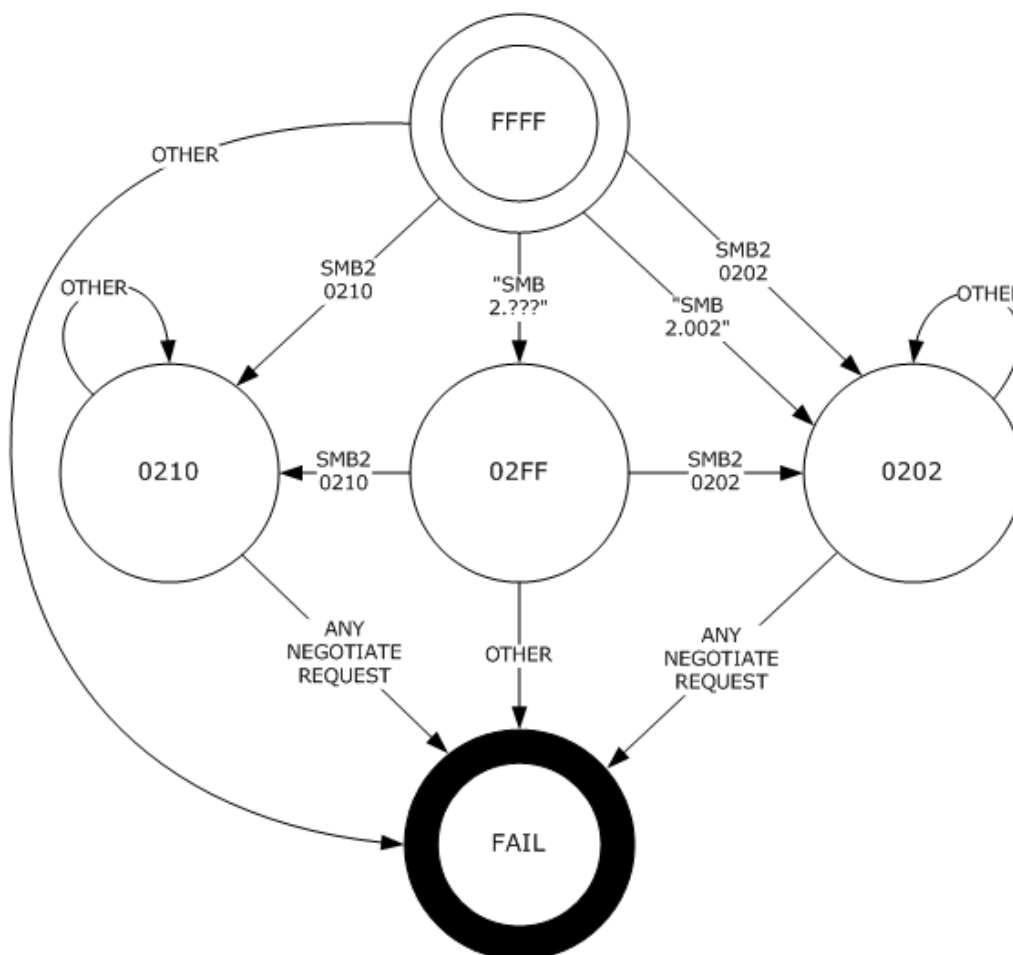


Figure 2: Connection.NegotiateDialect State Transitions in SMB 2.1 Server

1.8 Vendor-Extensible Fields

There are no vendor-extensible fields for the Server Message Block (SMB) Version 2 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 Protocol messages are encapsulated on the wire and common SMB 2 Protocol data types.

2.1 Transport

The Server Message Block (SMB) Version 2 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 Protocol is negotiated on the connection, there is no inheritance of the base SMB Protocol state. The SMB 2 Protocol takes over the transport connection that is initially used for negotiation, and thereafter, all protocol flow on that connection MUST be SMB 2 Protocol. The server assigns an implementation-specific name to each transport, as specified in [\[MS-SRVS\]](#) section 2.2.4.96.

2.2 Message Syntax

The SMB 2 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)

The SMB 2.1 dialect in the SMB 2 Protocol enhances the following categories of messages in the SMB 2 Protocol:

- Protocol Negotiation (SMB2 NEGOTIATE)
- Share Access (SMB2 TREE_CONNECT)
- File Access (SMB2 CREATE, SMB2 WRITE)
- Cache Coherency (SMB2 OPLOCK_BREAK)
- Hash Retrieval (SMB2 IOCTL)

This document specifies the messages in the preceding lists.

An SMB 2 Protocol message is the payload packet encapsulated in a transport packet.

All SMB 2 Protocol messages begin with a fixed-length [SMB2 header](#) that is described in section [2.2.1](#). The SMB2 header contains a **Command** field indicating the operation code that is requested by the client or responded to by the server. An SMB 2 Protocol message is of variable length, depending on the **Command** field in the SMB2 header and on whether the SMB 2 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 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 Unicode Transformation Format (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 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. More information about processing specific to printers is specified in 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 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 [3.2.4.24](#). The [SMB2 CANCEL Request](#) uses this format for canceling 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.

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	1	2	3	4	5	6	7	8	9	30	1
ProtocolId																															
StructureSize																CreditCharge															
Status																															
Command																CreditRequest/CreditResponse															
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.

CreditCharge (2 bytes): In the SMB 2.002 dialect, this field MUST NOT be used and MUST be reserved. The sender MUST set this to 0, and the receiver MUST ignore it. In the SMB 2.1 dialect, this field indicates the number of credits that this request consumes.

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
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/CreditResponse (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.

Flags (4 bytes): A flags field, which indicates how to process the operation. 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	When set, indicates that this is an ASYNC SMB2 header.

Value	Meaning
0x00000002	Always set for headers of the form described in this section.
SMB2_FLAGS_RELATED_OPERATIONS 0x00000004	When set in an SMB2 request, indicates that this request is a related operation in a compounded request chain. The use of this flag in an SMB2 request is as specified in 3.2.4.1.4 . When set in an SMB2 response, indicates that the request corresponding to this response was part of a related operation in a compounded request chain. The use of this flag in an SMB2 response is as specified in 3.3.5.2.6.2 .
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 0x10000000	When set, indicates that this command is a Distributed File 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 SMB2 header to the start of the subsequent 8-byte aligned SMB2 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 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 Protocol commands do not require the **SessionId** to be set to a nonzero value received from a previous [SMB2 SESSION SETUP Response](#). **SessionId** SHOULD [<2>](#) be set to 0 for the following commands:

- [SMB2 NEGOTIATE request](#)
- [SMB2 NEGOTIATE response](#)
- [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 SMB2 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.

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	1	2	3	4	5	6	7	8	9	30	1
ProtocolId																															
StructureSize																CreditCharge															
Status																															
Command																CreditRequest/CreditResponse															
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.

CreditCharge (2 bytes): In the SMB 2.002 dialect, this field MUST NOT be used and MUST be reserved. The sender MUST set this to 0, and the receiver MUST ignore it. In the SMB 2.1 dialect, this field indicates the number of credits that this request consumes.

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
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/CreditResponse (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.

Flags (4 bytes): A **Flags** field indicates how to process the operation. 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	When set, indicates that this is an ASYNC SMB2 header.

Value	Meaning
0x00000002	This flag MUST NOT be set when using the SYNC SMB2 header.
SMB2_FLAGS_RELATED_OPERATIONS 0x00000004	When set in an SMB2 request, indicates that this request is a related operation in a compounded request chain. The use of this flag in an SMB2 request is as specified in 3.2.4.1.4 . When set in an SMB2 response, indicates that the request corresponding to this response was part of a related operation in a compounded request chain. The use of this flag in an SMB2 response is as specified in 3.3.5.2.6.2 .
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 0x10000000	When set, indicates that this command is a 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 SMB2 header to the start of the subsequent 8-byte aligned SMB2 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 Protocol transport connection.

ProcessId (4 bytes): The client-side identification of the process that issued the request. The client MUST set this field to 0xFEFF. The server MUST set this field to the **ProcessId** value received in the corresponding request, if any, or to 0 otherwise. The client MUST ignore this field on receipt.

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 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](#)
- [SMB2 CANCEL Request](#)

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 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 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 SMB2 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.

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
StructureSize																Reserved															
ByteCount																															
ErrorData (variable)																															
...																															

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): This field MUST NOT be used and MUST be 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. [<3>](#)

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 requires 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. <4> The structure has the following format.

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
SymLinkLength																															
SymLinkErrorTag																															
ReparseTag																															
ReparseDataLength																UnparsedPathLength															
SubstituteNameOffset																SubstituteNameLength															
PrintNameOffset																PrintNameLength															
Flags																															
PathBuffer (variable)																															
...																															

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 **UnparsedPathLength**. 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**.)

UnparsedPathLength (2 bytes): The length, in bytes, of the unparsed portion of the path. The unparsed portion is the remaining part of the path after the symbolic link. See section 2.2.2.1.1 for examples.

SubstituteNameOffset (2 bytes): The offset, in bytes, from the beginning of the **PathBuffer** field, at which the substitute name is located. The substitute name is the name the client MUST use to access this file if it requires to follow the symbolic link.

SubstituteNameLength (2 bytes): The length, in bytes, of the substitute name string. If there is a terminating null character 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 **PathBuffer** field, at which the print name is located. 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 terminating null character 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): A 32-bit bit field that specifies whether the substitute is an absolute target path name or a path name relative to the directory containing the symbolic link.

This field contains one of the values in the table below.

Value	Meaning
0x00000000	The substitute name is an absolute target path name.
SYMLINK_FLAG_RELATIVE 0x00000001	When this Flags value is set, the substitute name is a path name relative to the directory containing the symbolic link.

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 operating system (OS).<5>
- For a relative target, the server MUST return a path that does not start with "\". The path MUST be evaluated, by the calling application, relative to the directory containing the symbolic link. The path can contain either "." to refer to the current directory or ".." to refer to the parent directory, and could contain multiple elements.

For more information on absolute and relative targets, see Handling the Symbolic Link Error Response (section [2.2.2.1.1](#)).

2.2.2.1.1 Handling the Symbolic Link Error Response

If a symbolic link error response is received, it MUST be processed by the calling application as follows:

1. The unparsed portion of the original path name that is located at the end of the path-name string MUST be extracted.

The size, in bytes, of the unparsed portion is specified in the **UnparsedPathLength** field. The byte count MUST be used from the end of the path-name string and walked backward to find the starting location of the unparsed bytes.

2. If the SYMLINK_FLAG_RELATIVE flag is not set in the **Flags** field of the symbolic link error response, the unparsed portion of the file name **MUST** be appended to the substitute name to create the new target path name.
3. If the SYMLINK_FLAG_RELATIVE flag is set in the **Flags** field of the symbolic link error response, the symbolic link name **MUST** be identified by backing up one path name element from the unparsed portion of the path name. The symbolic link **MUST** be replaced with the substitute name to create the new target path name.

The following clarifies handling of the symbolic link error response:

- An absolute symbolic link located on the server links "\\MachX\ShareY\Public\ProtocolDocs" to "\\??\D:\DonHall\MiscDocuments\PDocs".
 1. The original open request is for "\\MachX\ShareY\Public\ProtocolDocs\DailyDocs\[MS-SMB].doc".
 2. The server returns a symbolic link error response with the following data:
 - **UnparsedPathLength** field value of 0x2E
 - **PathBuffer** containing the Unicode string substitute name and print name "\\??\D:\DonHall\MiscDocuments\PDocsD:\DonHall\MiscDocuments\PDocs"
 - **SubstituteNameoffset** 0x00
 - **SubstituteNamelength** 0x44

The unparsed portion of the path name will be "\\DailyDocs\[MS-SMB].doc". Appending the substitute name with the unparsed portion of the file name gives the new target path name of "\\??\D:\DonHall\MiscDocuments\PDocs\DailyDocs\[MS-SMB].doc".

- A relative symbolic link located on the server links "\\MachX\ShareY\Public\ProtocolDocs" to "..\DonHall\Documents\PDocs".
 1. The original open request is for "\\MachX\ShareY\Public\ProtocolDocs\DailyDocs\[MS-SMB].doc".
 2. The server returns a symbolic link error response with the following data:
 - **UnparsedPathLength** field value of 0x2E
 - **PathBuffer** containing the Unicode string substitute name and print name "..\DonHall\Documents\PDocs..\DonHall\Documents\PDocs"
 - **SubstituteNameoffset** 0x00
 - **SubstituteNamelength** 0x34

The symbolic link name in this case is "ProtocolDocs".

Replacing the symbolic link name "ProtocolDocs" in the original open request (path name "\\MachX\ShareY\Public\ProtocolDocs\DailyDocs\[MS-SMB].doc") with the substitute name "..\DonHall\Documents\PDocs" gives the new target path name "\\MachX\ShareY\Public\..\DonHall\Documents\PDocs\DailyDocs\[MS-SMB].doc". Because "." and ".." are not permitted as components of a path name to be sent over the wire, before reissuing the SMB2 CREATE request the client **MUST** first eliminate the ".." by normalizing the

new target path name to "\\MachX\ShareY\DonHall\Documents\PDocs\DailyDocs\[MS-SMB].doc".

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 Protocol the client understands. This request is composed of an [SMB2 header](#), as specified in section [2.2.1](#), followed by this request structure.

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
StructureSize																DialectCount															
SecurityMode																Reserved															
Capabilities																															
ClientGuid																															
...																															
...																															
...																															
ClientStartTime																															
...																															
Dialects (variable)																															
...																															

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. [<6>](#)

SecurityMode (2 bytes): The security mode field specifies whether SMB signing is enabled, required at the server, or both. This 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. The client MUST set this bit if the SMB2_NEGOTIATE_SIGNING_REQUIRED bit is not set, and MUST NOT set this bit if the SMB2_NEGOTIATE_SIGNING_REQUIRED bit is set. The

Value	Meaning
	server MUST ignore this bit.
SMB2_NEGOTIATE_SIGNING_REQUIRED 0x0002	When set, indicates that security signatures are required by the client.

Reserved (2 bytes): This field MUST NOT be used and MUST be reserved. The client MUST set this to 0, and the server MUST ignore it on receipt.

Capabilities (4 bytes): This field MUST be set to 0 and the server MUST ignore it on receipt.

ClientGuid (16 bytes): It MUST be a **GUID** (as specified in [MS-DTYP] section 2.3.2.2) generated by the client, if the client implements SMB 2.1 dialect. Otherwise, the client MUST set this to 0.

ClientStartTime (8 bytes): This field MUST NOT be used and MUST be reserved. The client MUST set this to 0, and the server MUST ignore it on receipt.

Dialects (variable): An array of one or more 16-bit integers specifying the supported dialect revision numbers. The array MUST contain at least one of the following values. <7>

Value	Meaning
0x0202	SMB 2.002 dialect revision number. <8>
0x0210	SMB 2.1 dialect revision number. <9>

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.

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	1	2	3	4	5	6	7	8	9	30	1
StructureSize																SecurityMode															
DialectRevision																Reserved															
ServerGuid																															
...																															
...																															
...																															
Capabilities																															

MaxTransactSize	
MaxReadSize	
MaxWriteSize	
SystemTime	
...	
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 specifies whether SMB signing is enabled, required at the server, or both. This 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. The server MUST set this bit, and the client MUST return STATUS_INVALID_NETWORK_RESPONSE if the flag is missing.
SMB2_NEGOTIATE_SIGNING_REQUIRED 0x0002	When set, indicates that security signatures are required by the server.

DialectRevision (2 bytes): The preferred common SMB 2 Protocol dialect number from the **Dialects** array that is sent in the [SMB2 NEGOTIATE Request](#) (SECTION 2.2.3) or the SMB2 wildcard revision number. The server SHOULD set this field to one of the following values.[<10>](#)

Value	Meaning
0x0202	SMB 2.002 dialect revision number. <11>
0x0210	SMB 2.1 dialect revision number. <12>

Value	Meaning
0x02FF	SMB2 wildcard revision number; indicates that the server supports SMB 2.1 or future dialect revisions and expects the client to send a subsequent SMB2 Negotiate request to negotiate the actual SMB 2 Protocol revision to be used. The wildcard revision number is sent only in response to a multi-protocol negotiate request with the "SMB 2.???" dialect string.<13>

Reserved (2 bytes): This field MUST NOT be used and MUST be reserved. The server SHOULD set this to 0, and the client MUST ignore it on receipt.<14>

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.<15>

Capabilities (4 bytes): The Capabilities field 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).
SMB2_GLOBAL_CAP_LEASING 0x00000002	When set, indicates that the server supports leasing. This flag MUST only be valid for the SMB 2.1 dialect.
SMB2_GLOBAL_CAP_LARGE_MTU 0x00000004	When set, indicates that the server supports multi-credit operations. This flag MUST only be valid for the SMB 2.1 dialect.

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.<16>

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 SMB2 header to the security buffer.

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

Reserved2 (4 bytes): This field MUST NOT be used and MUST be 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 SHOULD contain a token as produced by the GSS protocol as specified in section 3.3.5.4. If **SecurityBufferLength** is 0, this field is empty and then client-initiated authentication, with an authentication protocol of the client's choice, will be used instead of server-initiated SPNEGO authentication as described in [MS-AUTHSO] section 3.2.2.

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 Protocol transport connection to the server. This request is composed of an [SMB2 header](#) as specified in section 2.2.1 followed by this request structure.

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
StructureSize																VcNumber								SecurityMode							
Capabilities																															
Channel																															
SecurityBufferOffset																SecurityBufferLength															
PreviousSessionId																															
...																															
Buffer (variable)																															
...																															

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 specifies whether SMB signing is enabled, required at the server, or both. This 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. The client MUST set this bit if the SMB2_NEGOTIATE_SIGNING_REQUIRED bit is not set, and MUST NOT set this bit if the SMB2_NEGOTIATE_SIGNING_REQUIRED bit is set. The server MUST ignore this bit.
SMB2_NEGOTIATE_SIGNING_REQUIRED	When set, indicates that security signatures are

Value	Meaning
0x02	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.
SMB2_GLOBAL_CAP_UNUSED2 0x00000004	SHOULD be set to zero and server MUST ignore.
SMB2_GLOBAL_CAP_UNUSED3 0x00000008	SHOULD be set to zero and server MUST ignore.

Values other than those that are defined in the previous table are unused at present and SHOULD [17](#) be treated as reserved.

Channel (4 bytes): This field MUST NOT be used and MUST be 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 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**. If the server initiated authentication using SPNEGO, the buffer MUST contain a token as produced by the GSS protocol as specified in section [3.2.4.2.3](#). If the client initiated authentication, see section [2.2.4](#), the buffer SHOULD [18](#) contain a token as produced by an authentication protocol of the client's choice.

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 [SMB2 header](#), as specified in section [2.2.1](#), that is followed by this response structure:

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

SecurityBufferOffset	SecurityBufferLength
Buffer (variable)	
...	

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 contain either 0 or one of 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 SMB2 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**. If the server initiated authentication using SPNEGO, the buffer MUST contain a token as produced by the GSS protocol as specified in section [3.3.5.5.3](#). If the client initiated authentication, see section [2.2.4](#), the buffer SHOULD [19](#) contain a token as produced by an authentication protocol of the client's choice.

2.2.7 SMB2 LOGOFF Request

The SMB2 LOGOFF Request packet is sent by the client to request termination of a particular session. This request is composed of an SMB2 header as specified in section [2.2.1](#) followed by this request structure.

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

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): This field MUST NOT be used and MUST be 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:

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

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): This field MUST NOT be used and MUST be 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.

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
StructureSize																Reserved															
PathOffset																PathLength															
Buffer (variable)																															
...																															

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): This field MUST NOT be used and MUST be 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 less than 256 characters in length, and it MUST be a NetBIOS name, a **fully qualified domain name (FQDN)**, or a textual IPv4 or IPv6 address. The share component of the path MUST be less than or equal to 80 characters in length. The share name MUST NOT contain any invalid characters, as specified in [\[MS-FSCC\]](#) section 2.1.6.<20>

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 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.

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				
StructureSize																ShareType								Reserved											
ShareFlags																																			
Capabilities																																			
MaximalAccess																																			

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): This field MUST NOT be used and MUST be 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 MUST 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,
SHI1005_FLAGS_FORCE_LEVELII_OPLOCK and SHI1005_FLAGS_ENABLE_HASH.

Descriptions of the individual flags follow.

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 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 Distributed File System (DFS) tree structure. The server SHOULD set the SHI1005_FLAGS_DFS bit in the ShareFlags field if the per-share property Share.IsDfs is TRUE.
SHI1005_FLAGS_DFS_ROOT 0x00000002	The specified share is present in a DFS tree structure. The server SHOULD set the SHI1005_FLAGS_DFS_ROOT bit in the ShareFlags field if the per-share property Share.IsDfs is TRUE.
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	The client MUST ignore this flag.
SHI1005_FLAGS_ACCESS_BASED_DIRECTORY_ENUM 0x00000800	The server will filter directory entries based on the access permissions of the client.
SHI1005_FLAGS_FORCE_LEVELII_OPLOCK 0x00001000	The server will not issue exclusive caching rights on this share. <21>
SHI1005_FLAGS_ENABLE_HASH 0x00002000	The share supports hash generation for branch cache retrieval of data. For more information, see section 2.2.31.2 .

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

Value	Meaning
SMB2_SHARE_CAP_DFS 0x00000008	The specified share is present in a DFS tree structure. The server MUST set the SMB2_SHARE_CAP_DFS bit in the Capabilities field if the per-

Value	Meaning
	share property Share.IsDfs is TRUE.

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 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.

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

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): This field MUST NOT be used and MUST be 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 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.

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

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): This field MUST NOT be used and MUST be 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.

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.
SMB2_OPLOCK_LEVEL_LEASE 0xFF	A lease is requested. If set, the request packet MUST contain an SMB2_CREATE_REQUEST_LEASE (section 2.2.13.2.8) create context. This value is only valid for the SMB 2.1 dialect.

ImpersonationLevel (4 bytes): This field specifies the impersonation level requested by the application that is issuing the create request, and MUST contain one of the following values. The server MUST validate this field, but otherwise ignore it.

Value	Meaning
Anonymous 0x00000000	The application-requested impersonation level is Anonymous.
Identification 0x00000001	The application-requested impersonation level is Identification.
Impersonation 0x00000002	The application-requested impersonation level is Impersonation.
Delegate 0x00000003	The application-requested impersonation level is Delegate.

Impersonation is specified in [\[MS-WSO\]](#) section 3.1.2.4; for more information about impersonation, see [\[MSDN-IMPERS\]](#).

SmbCreateFlags (8 bytes): This field MUST NOT be used and MUST be reserved. The client sets this to any value, and the server MUST ignore it on receipt.

Reserved (8 bytes): This field MUST NOT be used and MUST be 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 required, as specified in section [2.2.13.1](#).

FileAttributes (4 bytes): This field MUST be a combination of the values specified in [\[MS-FSCC\]](#) section 2.6, and MUST NOT include any values other than those specified in that section. This field is only valid when a file is created.

When a client requests that a file be created or opened, if any bits other than those documented in [\[MS-FSCC\]](#) section 2.6 are set, the server MUST fail the request with STATUS_INVALID_PARAMETER.

The server MUST ignore the **FileAttributes** field when opening an existing file, and the client SHOULD [<23>](#) set the **FileAttributes** field to 0.

The server MUST ignore **FileAttributes** for pipe opens.

For print files, if **FileAttributes** includes FILE_ATTRIBUTE_DIRECTORY the server MUST fail the open with the error code STATUS_NOT_SUPPORTED. 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 [<24>](#) ignore these values. The field MUST be constructed using a combination of zero or more of the following bit 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 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. <25>
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. <26>
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. <27>

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. [<28>](#)

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, FILE_OPEN_IF, or FILE_OPEN. With this flag, only the following CreateOptions values are valid: FILE_WRITE_THROUGH, and FILE_OPEN_FOR_BACKUP_INTENT. If the file being created or opened already exists and is not a directory file and FILE_CREATE is specified in the CreateDisposition field, then the server MUST fail the request with STATUS_OBJECT_NAME_COLLISION. If the file being created or opened already exists and is not a directory file and FILE_CREATE is not specified in the CreateDisposition field, then the server MUST fail the request with STATUS_NOT_A_DIRECTORY. The server MUST fail an invalid CreateDisposition field or an invalid combination of CreateOptions flags with STATUS_INVALID_PARAMETER.
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	This indicates that the application intends to read or write at sequential offsets using this handle, so the server SHOULD optimize for sequential access. However, the server MUST accept any access pattern. This flag value is incompatible with the FILE_RANDOM_ACCESS value.
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_SYNCHRONOUS_IO_ALERT 0x00000010	This bit SHOULD be set to 0 and MUST be ignored by the server. <29>
FILE_SYNCHRONOUS_IO_NONALERT 0x00000020	This bit SHOULD be set to 0 and MUST be ignored by the server. <30>
FILE_NON_DIRECTORY_FILE 0x00000040	The file being opened MUST NOT be a directory file or the server MUST fail the request with STATUS_FILE_IS_A_DIRECTORY. This flag MUST NOT be used with FILE_DIRECTORY_FILE or the server MUST fail the request with STATUS_INVALID_PARAMETER.
FILE_COMPLETE_IF_OPLOCKED 0x00000100	This bit SHOULD be set to 0 and MUST be ignored by the server. <31>
FILE_NO_EA_KNOWLEDGE 0x00000200	The caller does not understand how to handle extended attributes. If the request includes an SMB2_CREATE_EA_BUFFER create context, then the server MUST fail this request with STATUS_ACCESS_DENIED. If extended attributes with the FILE_NEED_EA flag (see [MS-FSCC] section 2.4.15) set

Value	Meaning
	are associated with the file being opened, then the server MUST fail this request with STATUS_ACCESS_DENIED.
FILE_RANDOM_ACCESS 0x00000800	This indicates that the application intends to read or write at random offsets using this handle, so the server SHOULD optimize for random access. However, the server MUST accept any access pattern. This flag value is incompatible with the FILE_SEQUENTIAL_ONLY value. If both FILE_RANDOM_ACCESS and FILE_SEQUENTIAL_ONLY are set, then FILE_SEQUENTIAL_ONLY is ignored.
FILE_DELETE_ON_CLOSE 0x00001000	The file MUST be automatically deleted when the last open request on this file is closed. When this option is set, the DesiredAccess field MUST include the DELETE flag. This option is often used for temporary files.
FILE_OPEN_BY_FILE_ID 0x00002000	This bit SHOULD be set to 0 and the server MUST fail the request with a STATUS_NOT_SUPPORTED error if this bit is set. <32>
FILE_OPEN_FOR_BACKUP_INTENT 0x00004000	The file is being opened for backup intent. That is, it is being opened or created for the purposes of either a backup or a restore operation. The server MAY check 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 check for access rights to the file before checking the DesiredAccess field.
FILE_NO_COMPRESSION 0x00008000	The file cannot be compressed.
FILE_RESERVE_OPFILTER 0x00100000	This bit SHOULD be set to 0 and the server MUST fail the request with a STATUS_NOT_SUPPORTED error if this bit is set. <33>
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.
FILE_OPEN_FOR_FREE_SPACE_QUERY 0x00800000	Open file to query for free space. The client SHOULD set this to 0 and the server MUST ignore it. <34>

NameOffset (2 bytes): The offset, in bytes, from the beginning of the SMB2 header to the 8-byte aligned file name. If SMB2_FLAGS_DFS_OPERATIONS is set in the Flags field of the SMB2 header, the file name can be prefixed with DFS link information that will be removed during DFS name normalization as specified in section [3.3.5.9](#). Otherwise, the file name is relative to the share that is identified by the TreeId in the SMB2 header. The **NameOffset** field SHOULD be set to the offset of the **Buffer** field from the beginning of the SMB2 header. The file name (after DFS normalization if needed) MUST conform to the specification of a relative pathname

in [\[MS-FSCC\]](#) section 2.1.5. A zero length file name indicates a request to open the root of the share.

NameLength (2 bytes): The length of the file name, in bytes. 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**. In the request, the **Buffer** field MUST be at least one byte in length. The file name (after DFS normalization if needed) MUST conform to the specification of a relative pathname in [\[MS-FSCC\]](#) section 2.1.5.

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

The following SMB2 Access Mask flag values can be used when accessing a file, pipe or printer.

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

Value	Meaning
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	SMB2 clients set this flag to any value. <35> SMB2 servers SHOULD <36> ignore this flag.
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] . <37>
MAXIMUM_ALLOWED 0x02000000	This value indicates that the client is requesting an open to the file 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 MAXIMUM_ALLOWED 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

The following SMB2 Access Mask flag values can be used when accessing a directory.

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
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.
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	SMB2 clients set this flag to any value. <38> SMB2 servers SHOULD <39> ignore this flag.
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]. <40>

Value	Meaning
MAXIMUM_ALLOWED 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 MAXIMUM_ALLOWED 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.

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
Next																															
NameOffset																NameLength															
Reserved																DataOffset															
DataLength																															
Buffer (variable)																															
...																															

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 "AISi"	The data contains the required allocation size of the newly created file.
SMB2_CREATE_QUERY_MAXIMAL_ACCESS_REQUEST "MxAc"	The client is requesting that the server return maximal access information.
SMB2_CREATE_TIMEWARP_TOKEN "TWrp"	The client is requesting that the server open an 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.
SMB2_CREATE_REQUEST_LEASE "RqLs"	The client is requesting that the server return a lease . This value is only supported for the SMB 2.1 dialect.

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

Reserved (2 bytes): This field **MUST NOT** be used and **MUST** be reserved. This value **MUST** be set to 0 by the client, and ignored by the server.

DataOffset (2 bytes): The offset, in bytes, from the beginning of this structure to the 8-byte aligned data payload. If DataLength is 0, the client **SHOULD** set this value to 0 and the server **MUST** ignore it on receipt. [<41>](#)

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. The server MUST ignore this Create Context for requests to open an existing file, a pipe, or a printer. 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. The server MUST ignore this Create Context for requests to open an existing file, a pipe, or a printer. The Data in the **Buffer** 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

When requesting a durable open, the client SHOULD also request a batch oplock (by setting RequestedOplockLevel to SMB2_OPLOCK_LEVEL_BATCH) or a handle caching lease (by using an SMB2_CREATE_REQUEST_LEASE Create Context with a LeaseState that includes SMB2_LEASE_HANDLE_CACHING). The server MUST ignore this Create Context if neither a batch oplock nor a handle caching lease is requested and granted. The format of the Data in the **Buffer** field of this SMB2_CREATE_CONTEXT MUST be as follows.

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

DurableRequest (16 bytes): A 16-byte field that MUST NOT be used and MUST be reserved. 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 [3.2.4.4](#).

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

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_REQUEST

The SMB2_CREATE_QUERY_MAXIMAL_ACCESS_REQUEST 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 in the **Buffer** field of the SMB2_CREATE_CONTEXT MUST either contain the following structure or be empty (0 bytes in length).

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

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 allocation size of a file that is being newly created or overwritten. The Data in the **Buffer** field of the SMB2_CREATE_CONTEXT MUST be as follows.

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

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 in the **Buffer** field of the SMB2_CREATE_CONTEXT MUST contain the following structure.

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

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.

2.2.13.2.8 SMB2_CREATE_REQUEST_LEASE

The SMB2_CREATE_REQUEST_LEASE context is specified on an [SMB2 CREATE Request](#) (section 2.2.13) packet when the client is requesting the server to return a lease. This is only valid for the SMB 2.1 dialect. The Data in the **Buffer** field of the [SMB2 CREATE CONTEXT](#) (section 2.2.13.2) structure MUST contain the following structure.

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

LeaseKey (16 bytes): A unique key that identifies the owner of the lease.

LeaseState (4 bytes): The requested lease state. This field MUST be constructed as a combination of the following values. [<42>](#)

Value	Meaning
SMB2_LEASE_NONE 0x00	No lease is requested.
SMB2_LEASE_READ_CACHING 0x01	A read caching lease is requested.
SMB2_LEASE_HANDLE_CACHING 0x02	A handle caching lease is requested.
SMB2_LEASE_WRITE_CACHING 0x04	A write caching lease is requested.

LeaseFlags (4 bytes): This field MUST NOT be used and MUST be reserved. The client MUST set this to 0, and the server MUST ignore it on receipt.

LeaseDuration (8 bytes): This field MUST NOT be used and MUST be reserved. The client MUST set this to 0, and the server MUST ignore it on receipt.

2.2.13.2.9 SMB2_CREATE_QUERY_ON_DISK_ID

The SMB2_CREATE_QUERY_ON_DISK_ID context is specified on an SMB2 CREATE Request (section [2.2.13](#)) when the client is requesting that the server return an identifier for the open file. The Data in the **Buffer** field of the SMB2_CREATE_CONTEXT MUST be empty.

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.

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
StructureSize																OplockLevel								Reserved							
CreateAction																															
CreationTime																															
...																															
LastAccessTime																															
...																															
LastWriteTime																															
...																															

ChangeTime
...
AllocationSize
...
EndOfFile
...
FileAttributes
Reserved2
FileId
...
...
...
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.[<43>](#43)

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.

Value	Meaning
SMB2_OPLOCK_LEVEL_BATCH 0x09	A batch oplock was granted.
OPLOCK_LEVEL_LEASE 0xFF	A lease is requested. If set, the response packet MUST contain an SMB2_CREATE_RESPONSE_LEASE create context.

Reserved (1 byte): This field MUST NOT be used and MUST be 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. [<44>](#)

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.
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): This field MUST NOT be used and MUST be reserved. The server SHOULD set this to 0, and the client MUST ignore it on receipt. [<45>](#)

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.

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

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](#)). If the open is a durable open, this value MUST be globally unique. If the open is not a durable open, this value MUST be unique for all persistent handles on that SMB2 transport connection.

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. This value MUST be unique for all volatile handles on the SMB2 transport connection.

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 Values, the format of the response is provided below.

2.2.14.2.1 SMB2_CREATE_EA_BUFFER

The SMB2_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, it MUST ignore this request.

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

Reserved (8 bytes): This field MUST NOT be used and MUST be reserved. The server MUST set this to 0, and the client MUST ignore the value on receipt.

2.2.14.2.4 SMB2_CREATE_DURABLE_HANDLE_RECONNECT

If the server implements SMB 2.1 and supports leasing, it MUST (in circumstances specified in section 3.3.5.9.7) respond to an [SMB2_CREATE_DURABLE_HANDLE_RECONNECT](#) request by returning an [SMB2_CREATE_RESPONSE_LEASE](#) create context. Otherwise, the server SHOULD NOT generate any create context response.

2.2.14.2.5 SMB2_CREATE_QUERY_MAXIMAL_ACCESS_RESPONSE

If the server attempts to query maximal access as part of processing a create request, it MUST return the results of the query to the client by including an SMB2_CREATE_QUERY_MAXIMAL_ACCESS_RESPONSE context in the response.

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

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 a 32-byte value that the client can use to identify the open file. The SMB2_CREATE_QUERY_ON_DISK_ID returns an SMB2_CREATE_CONTEXT in the response with the Name that is identified by SMB2_CREATE_QUERY_ON_DISK_ID as specified in section [2.2.13.2](#).

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

DiskIDBuffer (32 bytes): A 32-byte value that the client can use to identify the open file.

2.2.14.2.9 SMB2_CREATE_RESPONSE_LEASE

The server responds with a lease that is granted for this open. The data in the **Buffer** field of the [SMB2_CREATE_CONTEXT](#) structure MUST contain the following structure.

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

LeaseFlags
LeaseDuration
...

LeaseKey (16 bytes): A unique key that identifies the owner of the lease.

LeaseState (4 bytes): The granted lease state. This field MUST be constructed using the following values.

Value	Meaning
SMB2_LEASE_NONE 0x00	No lease is granted.
SMB2_LEASE_READ_CACHING 0x01	A read caching lease is granted.
SMB2_LEASE_HANDLE_CACHING 0x02	A handle caching lease is granted.
SMB2_LEASE_WRITE_CACHING 0x04	A write caching lease is granted.

LeaseFlags (4 bytes): If the server implements SMB 2.1, this field MUST be set to 0 or the following value. Otherwise, it is unused and MUST be reserved; the server MUST set this to 0, and the client MUST ignore it on receipt.

Value	Meaning
SMB2_LEASE_FLAG_BREAK_IN_PROGRESS 0x02	A break for the lease identified by the lease key is in progress.

LeaseDuration (8 bytes): This field MUST NOT be used and MUST be reserved. The server MUST set this to 0, and the client MUST ignore it on receipt.

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:

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
StructureSize																Flags															
Reserved																															
FileId																															

...
...
...

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 indicates how to process the operation. This field MUST be constructed using the following value:

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): This field MUST NOT be used and MUST be 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:

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
StructureSize																Flags															
Reserved																															
CreationTime																															
...																															
LastAccessTime																															
...																															
LastWriteTime																															
...																															

ChangeTime
...
AllocationSize
...
EndOfFile
...
FileAttributes

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 indicates how to process the operation. This field MUST be the logical OR of the following value, or zero if none are selected:

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): This field MUST NOT be used and MUST be 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:

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
StructureSize																Reserved1															
Reserved2																															
FileId																															
...																															
...																															
...																															

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

Reserved1 (2 bytes): This field MUST NOT be used and MUST be reserved. The client MUST set this to 0, and the server MUST ignore it on receipt.

Reserved2 (4 bytes): This field MUST NOT be used and MUST be 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:

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

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): This field MUST NOT be used and MUST be 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:

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
StructureSize																Padding								Reserved							
Length																															
Offset																															
...																															
FileId																															
...																															
...																															
...																															
MinimumCount																															
Channel																															
RemainingBytes																															
ReadChannelInfoOffset																ReadChannelInfoLength															

Buffer (variable)
...

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 from the start of the SMB2 header, in bytes, at which 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): This field MUST NOT be used and MUST be 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): This field MUST NOT be used and MUST be 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): This field MUST NOT be used and MUST be reserved. The client MUST set this field to 0, and the server MUST ignore it on receipt.

ReadChannelInfoLength (2 bytes): This field MUST NOT be used and MUST be 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:

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
StructureSize																DataOffset								Reserved							
DataLength																															
DataRemaining																															
Reserved2																															
Buffer (variable)																															
...																															

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): This field MUST NOT be used and MUST be 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): This field MUST NOT be used and MUST be reserved. The server MUST set this field to 0, and the client MUST ignore it on receipt.

Reserved2 (4 bytes): This field MUST NOT be used and MUST be 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 underlying object store, 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:

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

Offset	
...	
FileId	
...	
...	
...	
Channel	
RemainingBytes	
WriteChannelInfoOffset	WriteChannelInfoLength
Flags	
Buffer (variable)	
...	

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.

DataOffset (2 bytes): The offset, in bytes, from the beginning of the SMB2 header to the data being written.

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. If the write 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 write.

Channel (4 bytes): This field MUST NOT be used and MUST be 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): This field MUST NOT be used and MUST be reserved. The client MUST set this field to 0, and the server MUST ignore it on receipt.

WriteChannelInfoLength (2 bytes): This field MUST NOT be used and MUST be reserved. The client MUST set this field to 0, and the server MUST ignore it on receipt.

Flags (4 bytes): A **Flags** field indicates how to process the operation. This field MUST be constructed using the following value:

Value	Meaning
SMB2_WRITEFLAG_WRITE_THROUGH 0x00000001	The write request should be written to persistent storage before the response is sent regardless of how the file was opened. This value is only supported for the SMB 2.1 dialect.

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:

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
StructureSize																Reserved															
Count																															
Remaining																															
WriteChannelInfoOffset																WriteChannelInfoLength															

StructureSize (2 bytes): The server MUST set this field to 17, the actual size of the response structure notwithstanding.

Reserved (2 bytes): This field MUST NOT be used and MUST be 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): This field MUST NOT be used and MUST be reserved. The server MUST set this to 0, and the client MUST ignore it on receipt.

WriteChannelInfoOffset (2 bytes): This field MUST NOT be used and MUST be reserved. The server MUST set this to 0, and the client MUST ignore it on receipt.

WriteChannelInfoLength (2 bytes): This field MUST NOT be used and MUST be reserved. The server MUST set this to 0, and the client MUST ignore it on receipt.

2.2.23 SMB2 OPLOCK_BREAK Notification

2.2.23.1 Oplock Break Notification

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:

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
StructureSize																OplockLevel								Reserved							
Reserved2																															
FileId																															
...																															
...																															
...																															

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. [<46>](#)

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): This field MUST NOT be used and MUST be reserved. The server MUST set this to 0, and the client MUST ignore it on receipt.

Reserved2 (4 bytes): This field MUST NOT be used and MUST be 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 on which the **oplock break** occurred.

2.2.23.2 Lease Break Notification

The SMB2 Lease Break Notification packet is sent by the server when the underlying object store indicates that a lease is being broken, representing a change in the lease state. This is only valid on the SMB 2.1 dialect. This response is composed of an SMB2 header, as specified in section [2.2.1](#), followed by this notification structure:

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
StructureSize																Reserved															
Flags																															
LeaseKey																															
...																															
...																															
...																															
CurrentLeaseState																															
NewLeaseState																															
BreakReason																															
AccessMaskHint																															
ShareMaskHint																															

StructureSize (2 bytes): The server MUST set this to 44, indicating the size of the response structure, not including the header.

Reserved (2 bytes): This field MUST NOT be used and MUST be reserved. The server MUST set this to 0, and the client MUST ignore it on receipt.

Flags (4 bytes): The field MUST be constructed using the following value.

Value	Meaning
SMB2_NOTIFY_BREAK_LEASE_FLAG_ACK_REQUIRED 0x01	If set, indicate a Lease Break Acknowledgment is required.

LeaseKey (16 bytes): A unique key which identifies the owner of the lease.

CurrentLeaseState (4 bytes): The current lease state of the open. This field MUST be constructed using the following values.

Value	Meaning
SMB2_LEASE_READ_CACHING 0x01	A read caching lease is granted.
SMB2_LEASE_HANDLE_CACHING 0x02	A handle caching lease is granted.
SMB2_LEASE_WRITE_CACHING 0x04	A write caching lease is granted.

NewLeaseState (4 bytes): The new lease state for the open. This field MUST be constructed using the SMB2_LEASE_NONE or above values.

BreakReason (4 bytes): This field MUST NOT be used and MUST be reserved. The server MUST set this to 0, and the client MUST ignore it on receipt.

AccessMaskHint (4 bytes): This field MUST NOT be used and MUST be reserved. The server MUST set this to 0, and the client MUST ignore it on receipt.

ShareMaskHint (4 bytes): This field MUST NOT be used and MUST be reserved. The server MUST set this to 0, and the client MUST ignore it on receipt.

2.2.24 SMB2 OPLOCK_BREAK Acknowledgment

2.2.24.1 Oplock Break Acknowledgment

The 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 transition to none. Thus, the client does not send a request to the server because there is no question how the transition was made.

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
StructureSize																OplockLevel								Reserved							
Reserved2																															
FileId																															
...																															
...																															
...																															

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. <47>

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): This field MUST NOT be used and MUST be reserved. The client MUST set this to 0, and the server MUST ignore it on receipt.

Reserved2 (4 bytes): This field MUST NOT be used and MUST be 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 identifier of the file or pipe on which the oplock break occurred.

2.2.24.2 Lease Break Acknowledgment

The SMB2 Lease Break Acknowledgment packet is sent by the client in response to an SMB2 [Lease Break Notification](#) packet sent by the server. This request is valid only in the SMB 2.1 dialect. The server responds to a **lease break** acknowledgment with an SMB2 [Lease Break Response](#).

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
StructureSize																Reserved															
Flags																															
LeaseKey																															
...																															
...																															
...																															
LeaseState																															
LeaseDuration																															

...

- StructureSize (2 bytes):** The client MUST set this to 36, indicating the size of the request structure, not including the header.
- Reserved (2 bytes):** This field MUST NOT be used and MUST be reserved. The client MUST set this to 0, and the server MUST ignore it on receipt.
- Flags (4 bytes):** This field MUST NOT be used and MUST be reserved. The client MUST set this to 0, and the server MUST ignore it on receipt.
- LeaseKey (16 bytes):** A unique key which identifies the owner of the lease.
- LeaseState (4 bytes):** The lease state in the Lease Break Acknowledgment message MUST be a subset of the lease state granted by the server via the preceding Lease Break Notification message. [<48>](#) This field MUST be constructed using the following values:

Value	Meaning
SMB2_LEASE_NONE 0x00	No lease is granted.
SMB2_LEASE_READ_CACHING 0x01	A read caching lease is accepted.
SMB2_LEASE_HANDLE_CACHING 0x02	A handle caching lease is accepted.
SMB2_LEASE_WRITE_CACHING 0x04	A write caching lease is accepted.

- LeaseDuration (8 bytes):** This field MUST NOT be used and MUST be reserved. The client MUST set this to 0, and the server MUST ignore it on receipt.

2.2.25 SMB2 OPLOCK_BREAK Response

2.2.25.1 Oplock Break Response

The 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:

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
StructureSize																OplockLevel								Reserved							
Reserved2																															
FileId																															
...																															

...
...

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 resulting oplock level. This MUST be the same as the level that is specified by the client in its oplock break acknowledgment packet. This field MUST contain one of the following values.

Value	Meaning
SMB2_OPLOCK_LEVEL_NONE 0x00	The server has lowered oplock level for this file to none.
SMB2_OPLOCK_LEVEL-II 0x01	The server has lowered oplock level for this file to level II.

Reserved (1 byte): This field MUST NOT be used and MUST be reserved. The server MUST set this to 0, and the client MUST ignore it on receipt.

Reserved2 (4 bytes): This field MUST NOT be used and MUST be 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 on which the oplock break occurred.

2.2.25.2 Lease Break Response

The SMB2 Lease Break Response packet is sent by the server in response to a [Lease Break Acknowledgment](#) from the client. This request is valid only in the SMB 2.1 dialect.

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
StructureSize																Reserved															
Flags																															
LeaseKey																															
...																															
...																															
...																															
LeaseState																															

LeaseDuration
...

StructureSize (2 bytes): The server MUST set this to 36, indicating the size of the response structure, not including the header.

Reserved (2 bytes): This field MUST NOT be used and MUST be reserved. The server MUST set this to 0, and the client MUST ignore it on receipt.

Flags (4 bytes): This field MUST NOT be used and MUST be reserved. The server MUST set this to 0, and the client MUST ignore it on receipt.

LeaseKey (16 bytes): A unique key which identifies the owner of the lease.

LeaseState (4 bytes): The requested lease state. This field MUST be constructed using the following values:

Value	Meaning
SMB2_LEASE_NONE 0x00	No lease is granted.
SMB2_LEASE_READ_CACHING 0x01	A read caching lease is granted.
SMB2_LEASE_HANDLE_CACHING 0x02	A handle caching lease is granted.
SMB2_LEASE_WRITE_CACHING 0x04	A write caching lease is granted.

LeaseDuration (8 bytes): This field MUST NOT be used and MUST be reserved. The server MUST set this to 0, and the client MUST ignore it on receipt.

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.

Byte range locks in SMB2 are associated with the handle (SMB2 **FileId**) on which the lock is taken. Read/write/lock requests using the same SMB2 **FileId** will not conflict. However, I/O on a different handle (a different SMB2 **FileId**) will conflict. It is the client's responsibility to locally resolve lock conflicts across multiple processes on the same client, if any such conflicts exist.

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
StructureSize																LockCount															
LockSequence																															

FileId
...
...
...
Locks (variable)
...

StructureSize (2 bytes): The client MUST set this to 48, indicating the size of an SMB2 LOCK Request with a single [SMB2_LOCK_ELEMENT](#) 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_ELEMENT structures that are contained in the **Locks[]** array. The lock count MUST be greater than or equal to 1.

LockSequence (4 bytes): In the SMB 2.002 dialect, this field is unused and MUST be reserved. The client MUST set this to 0, and the server MUST ignore it on receipt. In the SMB 2.1 dialect, this field indicates a value that identifies a lock or unlock request uniquely across all lock or unlock requests that are sent on the same file.

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.

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

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 to process the operation. 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. All 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): This field MUST NOT be used and MUST be 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:

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

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

Reserved (2 bytes): This field MUST NOT be used and MUST be 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:

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

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

Reserved (2 bytes): This field MUST NOT be used and MUST be 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 is sent by the server to confirm that an SMB2 ECHO Request (section [2.2.28](#)) was successfully processed. This response is composed of an [SMB2 header](#), as specified in section [2.2.1](#), followed by the following response structure.

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

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

Reserved (2 bytes): This field MUST NOT be used and MUST be 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:

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

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): This field MUST NOT be used and MUST be 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 [3.2.4.20](#) 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.

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
StructureSize																Reserved															
CtlCode																															
FileId																															
...																															
...																															
...																															
InputOffset																															
InputCount																															
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): This field MUST NOT be used and MUST be 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 [3.2.4.20](#) and [3.3.5.15](#).

Name	Value
FSCTL_DFS_GET_REFERRALS	0x00060194
FSCTL_PIPE_PEEK	0x0011400C
FSCTL_PIPE_WAIT	0x00110018
FSCTL_PIPE_TRANSCEIVE	0x0011C017
FSCTL_SRV_COPYCHUNK	0x001440F2
FSCTL_SRV_ENUMERATE_SNAPSHOTS	0x00144064
FSCTL_SRV_REQUEST_RESUME_KEY	0x00140078
FSCTL_SRV_READ_HASH	0x001441bb
FSCTL_SRV_COPYCHUNK_WRITE	0x001480F2
FSCTL_LMR_REQUEST_RESILIENCY	0x001401D4

FSCTL_PIPE_TRANSCEIVE is valid only on a named pipe with mode set to FILE_PIPE_MESSAGE_MODE as specified in [\[MS-FSCC\]](#) section 2.4.29.

FSCTL_SRV_COPYCHUNK and FSCTL_SRV_COPYCHUNK_WRITE FSCTL codes are used for performing server side copy operations. These FSCTLs are issued by the application against an open handle to the target file. FSCTL_SRV_COPYCHUNK is issued when a handle has FILE_READ_DATA and FILE_WRITE_DATA access to the file; FSCTL_SRV_COPYCHUNK_WRITE is issued when a handle only has FILE_WRITE_DATA access.

FileId (16 bytes): An [SMB2_FILEID](#) identifier of the file on which to perform the command. For FSCTL_DFS_GET_REFERRALS and FSCTL_PIPE_WAIT **CtlCode** values, this parameter MUST be set to -1.

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.[<49>](#)

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 [SMB2 IOCTL Response](#).

OutputOffset (4 bytes): The client SHOULD set this to 0.[<50>](#)

OutputCount (4 bytes): The client MUST set this to 0.

MaxOutputResponse (4 bytes): The maximum number of bytes that the server can return for the output data in the SMB2 IOCTL Response.

Flags (4 bytes): A **Flags** field indicating how to process the operation. This field **MUST** be constructed using one of the following values.

Value	Meaning
0x00000000	If Flags is set to this value, the request is an IOCTL request.
SMB2_0_IOCTL_IS_FSCTL 0x00000001	If Flags is set to this value, the request is an FSCTL request.

Reserved2 (4 bytes): This field **MUST NOT** be used and **MUST** be 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 or FSCTL_SRV_COPYCHUNK_WRITE, 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	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
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 NOT be used and MUST be reserved. 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:

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

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.

Reserved (4 bytes): this field MUST NOT be used and MUST be reserved. SHOULD be set to zero and MUST be ignored.[<51>](#)

2.2.31.2 SRV_READ_HASH Request

The SRV_READ_HASH request is sent to the server by the client in an **SMB2 IOCTL Request** FSCTL_SRV_READ_HASH to retrieve data from the **Content Information File** associated with a specified file. The request is valid only for the SMB 2.1 dialect. 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	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
HashType																															
HashVersion																															
HashRetrievalType																															
Length																															
Offset																															

HashType (4 bytes): The hash type of the request indicates what the hash is used for. This field **MUST** be set to the following value:

Value	Meaning
SRV_HASH_TYPE_PEER_DIST 0x00000001	Indicates the hash is requested for branch caching as described in [MS-PCCRC] .

HashVersion (4 bytes): The version number of the algorithm used to create the Content Information. This field **MUST** be set to the following value:

Value	Meaning
SRV_HASH_VER_1 0x00000001	Branch cache version 1.

HashRetrievalType (4 bytes): Indicates the nature of the **Offset** field. This field **MUST** be set to the following value:

Value	Meaning
SRV_HASH_RETRIEVE_HASH_BASED 0x00000001	The Offset field in the SRV_READ_HASH request is relative to the beginning of the Content Information File.

Length (4 bytes): The maximum length, in bytes, of the data to be returned in the SRV_READ_HASH response to the client.

Offset (4 bytes): This value is the offset of the data to be retrieved, in bytes, from the beginning of the Content Information File.

2.2.31.2.1 HASH_HEADER

All Content Information Files **MUST** start with a valid format HASH_HEADER as follows. The format is valid only for the SMB 2.1 dialect.

Content Information follows this header at an offset indicated by the **HashBlobOffset** field, and takes the form of a Content Information data structure as specified in [\[MS-PCCRC\]](#) section 2.3.

0	1	2	3	4	5	6	7	8	9	0	1	2	3	4	5	6	7	8	9	0	1	2	3	4	5	6	7	8	9	0	1
HashType																															
HashVersion																															
SourceFileChangeTime																															
...																															
SourceFileSize																															
...																															
HashBlobLength																															
HashBlobOffset																															
Dirty																SourceFileNameLength															
SourceFileName (variable)																															
...																															

HashType (4 bytes): The hash type of the request indicates what the hash is used for. This field **MUST** be constructed using the following value.

Value	Meaning
SRV_HASH_TYPE_PEER_DIST 0x00000001	Indicates that the hash is requested for branch caching as described in [MS-PCCRC].

HashVersion (4 bytes): The version number of the algorithm that is used to create the Content Information algorithm. This field **MUST** be constructed using the following value.

Value	Meaning
SRV_HASH_VER_1 0x00000001	Branch cache version 1.

SourceFileChangeTime (8 bytes): The last update time for the source file from which the Content Information is generated, in [FILETIME](#) format as specified in [\[MS-DTYP\]](#) section 2.3.1.

SourceFileSize (8 bytes): The length, in bytes, of the source file from which the Content Information is generated.

HashBlobLength (4 bytes): The length, in bytes, of the Content Information.

HashBlobOffset (4 bytes): The offset of the Content Information, in bytes, from the beginning of the Content Information File.

Dirty (2 bytes): A flag that indicates whether the Content Information File is currently being updated. A nonzero value indicates TRUE.

SourceFileNameLength (2 bytes): The length, in bytes, of the source file full name.

SourceFileName (variable): A variable-length buffer that contains the source file full name, with length indicated by **SourceFileNameLength**.<52>

2.2.31.3 NETWORK_RESILIENCY_REQUEST Request

The NETWORK_RESILIENCY_REQUEST request packet is sent to the server by the client in an [SMB2 IOCTL Request \(section 2.2.31\)](#) FSCTL_LMR_REQUEST_RESILIENCY to request resiliency for a specified open file. The request is valid only for the SMB 2.1 dialect. 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	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
Timeout																															
Reserved																															

Timeout (4 bytes): The requested time the server should hold the file open after a disconnect before releasing it. This time is in milliseconds.

Reserved (4 bytes): This field MUST NOT be used and MUST be reserved. The client MUST set this to 0, and the server MUST ignore it on receipt.

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:

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

InputOffset
InputCount
OutputOffset
OutputCount
Flags
Reserved2
Buffer (variable)
...

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): This field MUST NOT be used and MUST be 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. SMB2-specific values are listed in section [2.2.31](#).

FileId (16 bytes): An [SMB2_FILEID](#) identifier of the file on which the command was performed. If the **CtlCode** field value is FSCTL_DFS_GET_REFERRALS or FSCTL_PIPE_WAIT, this field MUST be set to -1 by the server and MUST be ignored by the client.

InputOffset (4 bytes): **InputOffset** SHOULD be set to the offset, in bytes, from the beginning of the SMB2 header to the **Buffer[]** field of the IOCTL response.

InputCount (4 bytes): **InputCount** SHOULD [<53>](#) be set to zero in the IOCTL response. An exception for pass-through operations is discussed in section [3.3.5.15.8](#).

OutputOffset (4 bytes): The offset, in bytes, from the beginning of the SMB2 header to the output data buffer. If output data is returned, the output offset MUST be set to **InputOffset** + **InputCount** rounded up to a multiple of 8. If no output data is returned for the FSCTL/IOCTL command that was issued, then this value SHOULD [<54>](#) be set to 0.

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

Flags (4 bytes): This field MUST NOT be used and MUST be reserved. The server MUST set this field to 0, and the client MUST ignore it on receipt.

Reserved2 (4 bytes): This field MUST NOT be used and MUST be 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:

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

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 indicates 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 indicates the maximum number of chunks that the server will accept in a single request. This would allow the client to correctly reissue the request.

ChunkBytesWritten (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 indicates 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 indicates 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 indicates 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 indicates 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 section [3.3.5.15.1](#). This packet MUST contain all the revision time-stamps that are associated with the Tree Connect share in which the open resides, provided that the buffer size required is less than or equal to the maximum output buffer size received in the SMB2 IOCTL request. This SRV_SNAPSHOT_ARRAY is placed in the **Buffer** field in the SMB2 IOCTL Response, [<55>](#) 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:

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

SnapshotArraySize
SnapShots (variable)
...

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. If the output buffer could not accommodate the entire array, NumberOfSnapShotsReturned will be zero.

SnapshotArraySize (4 bytes): The length, in bytes, of the SnapShots[] array. If the output buffer is too small to accommodate the entire array, SnapshotArraySize will be the amount of space that the array would have occupied.

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. It will be empty if the output buffer could not accommodate the entire array.

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:

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
ResumeKey																															
...																															
...																															
...																															
...																															
...																															
ContextLength																															
Context (variable)																															
...																															

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** or **FSCTL_SRV_COPYCHUNK_WRITE** 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 MUST set this field to 0, and the client MUST ignore it on receipt.

Context (variable): The context extended information. [<56>](#)

2.2.32.4 SRV_READ_HASH Response

The SRV_READ_HASH response is returned to the client by the server in an [SMB2_IOCTL Response](#) for the FSCTL_SRV_READ_HASH request. This structure 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](#). The SRV_READ_HASH response MUST be formatted as follows:

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
Offset																															
...																															
BufferLength																															
Reserved																															
Buffer (variable)																															
...																															

Offset (8 bytes): The offset, in bytes, from the beginning of the Content Information File to the portion retrieved. This is equal to the **Offset** field in the [SRV_READ_HASH request](#).

BufferLength (4 bytes): The length, in bytes, of the retrieved portion of the Content Information File.

Reserved (4 bytes): This field MUST NOT be used and MUST be 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 retrieved portion of the Content Information File.

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:

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
StructureSize																FileInformationClass								Flags							
FileIndex																															
FileId																															
...																															
...																															
...																															
FileNameOffset																FileNameLength															
OutputBufferLength																															
Buffer (variable)																															
...																															

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 number assigned by the underlying object store that uniquely identifies a file within a volume.
FileBothDirectoryInformation 0x03	Basic information plus extended attribute size and short name about a file or directory.
FileIdBothDirectoryInformation 0x25	FileBothDirectoryInformation plus volume file ID about a file or directory.
FileNamesInformation 0x0C	Detailed information on the names of files and directories in a directory.

Flags (1 byte): Flags indicating how the query directory operation MUST be processed. This field MUST be a logical OR of the following values, or zero if none are selected:

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<57> return entries beginning at the byte number specified by FileIndex .
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.

SMB2_REOPEN implies SMB2_RESTART_SCANS as well.

FileIndex (4 bytes): The byte offset within the directory, indicating the position at which to resume the enumeration. If SMB2_INDEX_SPECIFIED is set in **Flags**, this value MUST be supplied, and should be based on the **FileIndex** value received in a previous enumeration response. Otherwise, it MUST be set to zero and the server MUST ignore it.

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 [\[MS-CIFS\]](#) section 2.2.1.1.3.<58>

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:

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

OutputBufferLength
Buffer (variable)
...

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:

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
StructureSize																Flags															
OutputBufferLength																															
FileId																															
...																															
...																															
...																															
CompletionFilter																															
Reserved																															

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 the logical OR of the following value, or zero if none is selected:

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 0x00000004	The client is notified if a file's attributes change. Possible 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): This field MUST NOT be used and MUST be 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:

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
StructureSize																OutputBufferOffset															
OutputBufferLength																															
Buffer (variable)																															
...																															

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:

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

FileNameLength
FileName (variable)
...

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:

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31								
StructureSize																InfoType								FileInfoClass															
OutputBufferLength																																							
InputBufferOffset																Reserved																							
InputBufferLength																																							
AdditionalInformation																																							
Flags																																							
FileId																																							
...																																							
...																																							
...																																							
Buffer (variable)																																							
...																																							

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 underlying object store information is requested.
SMB2_0_INFO_SECURITY 0x03	The security information is requested.
SMB2_0_INFO_QUOTA 0x04	The underlying object store 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:

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

For underlying object store 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:

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

For security queries, this field MUST be set to 0. For quota queries, this field SHOULD [<59>](#) 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, in bytes, from the beginning of the SMB2 header to the input buffer. For quota requests, the input buffer MUST contain an [SMB2_QUERY_QUOTA_INFO](#), as specified in section [2.2.37.1](#). For FileFullEaInformation requests, the input buffer MUST contain the user supplied EA list with zero or more FILE_GET_EA_INFORMATION structures, specified in [\[MS-FSCC\]](#) section 2.4.15.1. For other information queries, this field SHOULD [<60>](#) be set to 0.

Reserved (2 bytes): This field MUST NOT be used and MUST be 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 FileFullEaInformation requests, this MUST be set to the length of the user supplied EA list specified in [\[MS-FSCC\]](#) section 2.4.15.1. 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 and the application has not provided a list of EAs to query, but has provided an index into the object's full extended attribute information array at which to start the query, this field MUST contain a ULONG representation of that index. For all other queries, this field MUST be set to 0 and the server MUST ignore it.

Flags (4 bytes): The flags MUST be set to a combination of zero or more of these bit values for a FileFullEaInformation query.

Value	Meaning
SL_RESTART_SCAN	Restart the scan for EAs from the beginning.

Value	Meaning
0x00000001	
SL_RETURN_SINGLE_ENTRY 0x00000002	Return a single EA entry in the response buffer.
SL_INDEX_SPECIFIED 0x00000004	The caller has specified an EA index.

For all other queries, 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 underlying object store 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. [<61>](#)

For quota requests, the input **Buffer** MUST contain an SMB2_QUERY_QUOTA_INFO, as specified in section [2.2.37.1](#). For a FileFullEaInformation query, the **Buffer** MUST be in one of the following formats:

- A zero-length buffer as indicated by an InputBufferLength that is equal to zero.
- A list of FILE_GET_EA_INFORMATION structures provided by the application, as specified in [\[MS-FSCC\]](#) section 2.4.15.1.

2.2.37.1 SMB2_QUERY_QUOTA_INFO

The SMB2_QUERY_QUOTA_INFO packet specifies the quota information to return.

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	1	2	3	4	5	6	7	8	9	30	1
ReturnSingle									RestartScan							Reserved															
SidListLength																															
StartSidLength																															
StartSidOffset																															
SidBuffer (variable)																															
...																															

ReturnSingle (1 byte): A Boolean value, where zero represents FALSE and nonzero represents TRUE. [<62>](#) If the **ReturnSingle** field is TRUE, the server MUST return a single value. Otherwise, the server SHOULD return the maximum number of entries that will fit in the maximum output size that is indicated in the request.

RestartScan (1 byte): A Boolean value, where zero represents FALSE and nonzero represents TRUE.<63> If **RestartScan** is TRUE, the quota information MUST be read from the beginning. Otherwise, the quota information MUST be continued from the previous enumeration that was executed on this open.

Reserved (2 bytes): This field MUST NOT be used and MUST be 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 **SidBuffer** when sent in format 1 as defined in the **SidBuffer** field or zero in all other cases.

StartSidLength (4 bytes): The length, in bytes, of the **SID**, as specified in [MS-DTYP] section 2.4.2.2, when sent in format 2 as defined in the **SidBuffer** field, or zero in all other cases.

StartSidOffset (4 bytes): The offset, in bytes, from the start of the **SidBuffer** field to the **SID** when sent in format 2 as defined in the **SidBuffer** field, or zero in all other cases.

SidBuffer (variable): If this field is empty, then **SidListLength**, **StartSidLength** and **StartSidOffset** MUST each be set to zero. If the field is not empty, then it MUST contain either one of the following two formats:

1. A list of FILE_GET_QUOTA_INFORMATION structures, as described in [MS-FSCC] section 2.4.33.1.
2. A SID.<64>

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.

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
StructureSize																OutputBufferOffset															
OutputBufferLength																															
Buffer (variable)																															
...																															

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	The value depends on FileInfoClass, as specified in section 2.2.37 .	See [MS-FSCC] section 2.4. For FileFullEaInformation, the server MUST return the list of extended attributes (EA) that will fit in the Buffer , beginning with the attribute whose index is specified by the AdditionalInformation field of the request. The size of the returned buffer is equal to the size of the EA entries that are returned.
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 2.4.33.

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 object store. This request consists of an [SMB2 header](#), as specified in section [2.2.1](#), followed by this request structure.

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31																								
StructureSize																InfoType								FileInfoClass																															
BufferLength																																																							
BufferOffset																Reserved																																							
AdditionalInformation																																																							
FileId																																																							
...																																																							

...
...
Buffer (variable)
...

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 as follows.

Value	Meaning
SMB2_0_INFO_FILE 0x01	The file information is being set.
SMB2_0_INFO_FILESYSTEM 0x02	The underlying object store information is being set.
SMB2_0_INFO_SECURITY 0x03	The security information is being set.
SMB2_0_INFO_QUOTA 0x04	The underlying object store quota information is being set.

FileInfoClass (1 byte): For setting file information, this field MUST contain one of the following FILE_INFORMATION_CLASS values, as specified in section [3.3.5.21.1](#) and [\[MS-FSCC\]](#) section 2.4:

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

For setting underlying object store 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): This field **MUST NOT** be used and **MUST** be 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 underlying object store 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:	The security descriptor data structure, as specified in [MS-

InfoType	AdditionalInformation	Buffer format specification
	OWNER_SECURITY_INFORMATION GROUP_SECURITY_INFORMATION LABEL_SECURITY_INFORMATION DACL_SECURITY_INFORMATION SACL_SECURITY_INFORMATION	DTYP section 2.4.6, populated with the data specified by the AdditionalInformation value.
SMB2_0_INFO_QUOTA	0	See [MS-FSCC] section 2.4.33.

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:

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

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 section describes a conceptual model of possible data organization that an implementation maintains to participate in this protocol. The described organization is provided to facilitate the explanation of how the protocol behaves. This document does not mandate that implementations adhere to this model as long as their external behavior is consistent with 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. [<65>](#)

3.1.4 Higher-Layer Triggered Events

3.1.4.1 Signing An Outgoing Message

If the client or server sending the message requires that the message be signed, it provides 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 a 32-byte hash using HMAC-SHA256 over the entire message, beginning with the SMB2 Header from step 1, and using the session key as the signing key. The HMAC-SHA256 hash is specified in [\[FIPS180-2\]](#) and [\[RFC2104\]](#). If the message is part of a compounded chain, any padding at the end of the message **MUST** be used in the hash computation.
3. The first 16 bytes (the high-order portion) of the hash created in step 2 **MUST** be copied (beginning with the first, most significant, byte) into the 16-byte signature field of the SMB2 Header.

Determining when a client will sign an outgoing message is specified in [3.2.4.1.1](#), and determining when a server will sign an outgoing message is specified in [3.3.4.1.1](#).

3.1.5 Processing Events and Sequencing Rules

None.

3.1.5.1 Verifying an Incoming Message

If a client or server requires verification of a signed message, it provides the message length, the buffer containing the message, and the session key. The following steps describe how the signature MUST be verified:

1. The receiver MUST save the 16-byte signature from the **Signature** field in the [SMB2 Header](#) for use in step 4.
2. The receiver MUST zero out the 16-byte signature field in the SMB2 Header of the incoming message.
3. The receiver MUST compute a 32-byte hash using HMAC-SHA256 over the entire message, beginning with the SMB2 Header from step 2, and using the session key as the signing key. The HMAC-SHA256 hash is specified in [\[FIPS180-2\]](#) and [\[RFC2104\]](#). If the message is part of a compounded chain, any padding at the end of the message MUST be used in the hash computation.
4. If the first 16 bytes (the high-order portion) of the computed signature from step 3 matches the saved signature from step 1, the message is signed correctly.

Determining when a client will verify a signature and the action taken on the result of verification is specified in section [3.2.5.1.2](#). Determining when a server will verify a signature and the action taken on the result of verification is specified in section [3.3.5.2.3](#).

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 Global

The client MUST implement the following:

ConnectionTable: A table of active SMB2 transport connections, as specified in section [3.2.1.2](#), that are established to remote servers, indexed by the **Connection.ServerName**.

If a client implements the SMB 2.100 dialect, it MUST also implement the following:

GlobalFileTable: A table of opened files, as specified in section [3.2.1.5](#), indexed by name, as specified in section [3.2.4.3](#), and also indexed by File.LeaseKey.

ClientGuid: A global identifier for this client.

3.2.1.2 Per SMB2 Transport Connection

The client MUST implement the following:

Connection.SessionTable: A table of authenticated sessions, as specified in section [3.2.1.3](#), 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, as specified in section [3.2.1.7](#), that have been issued on this connection and are awaiting a response. The table MUST allow lookup by **Request.CancelId** and by **MessageId**, and each request MUST store the time at which the request was sent.

Connection.SequenceWindow: A table of available **sequence numbers** for sending requests to the server, as specified in section [3.2.4.1.6](#).

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. [<66>](#)

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 requests/responses on this connection to be signed.

Connection.ServerName: A null-terminated Unicode UTF-16 fully qualified domain name, a NetBIOS name, or an IP address of the server machine.

If a client implements the SMB 2.1 dialect, it MUST also implement the following:

- **Connection.Dialect:** The dialect of SMB2 negotiated with the server. This value MUST be either "2.002", "2.100", or "Unknown".
- **Connection.SupportsLeasing:** A Boolean indicating whether the server supports leasing functionality.

- **Connection.SupportsMultiCredit:** A Boolean indicating whether the server supports multi-credit operations.

3.2.1.3 Per Session

The client MUST implement the following:

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 [3.2.1.4](#). The table MUST allow lookup by both **TreeConnect.TreeConnectId** and 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.UserCredentials: An opaque implementation-specific entity that identifies the credentials that were used to authenticate to the server.

3.2.1.4 Per Tree Connect

The client MUST implement the following:

TreeConnect.ShareName: The share name corresponding to this tree connect.

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

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

TreeConnect.IsDfsShare: A Boolean that, if set, indicates that the tree connect was established to a DFS share.

3.2.1.5 Per Open File

If a client implements the SMB 2.1 dialect, then for each opened file (distinguished by name, as specified in section [3.2.4.3](#)), the client MUST implement the following:

- **File.OpenTable:** A table of **Opens** to this file.
- **File.LeaseKey:** A 128-bit key generated by the client, which uniquely identifies this file's entry in the **GlobalFileTable**.
- **File.LeaseState:** The lease level state granted for this file by the server as described in [2.2.13.2.8](#).

A lease state containing SMB2_LEASE_WRITE_CACHING implies that the client has exclusive access to the file and it MAY choose to cache writes to the file. The client MAY also choose to cache byte-range locks.

A lease state containing SMB2_LEASE_READ_CACHING implies there might be multiple readers of the file, and the client MAY choose to satisfy reads from its cache. The client MUST send all writes to the server.

A lease state containing SMB2_LEASE_HANDLE_CACHING implies that the client MAY choose to keep open handles to the file even after the application that opened the file has closed its handles or has ended.

3.2.1.6 Per Application Open of a File

The client MUST implement the following:

Open.FileId: The [SMB2_FILEID](#), as specified in section [2.2.14.1](#), returned by the server for this open.

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.

Open.ResilientHandle: A Boolean that indicates whether resiliency was granted for this open by the server.

Open.LastDisconnectTime: The time at which the last network disconnect occurred on the connection used by this open.

Open.ResilientTimeout: The minimum time (in milliseconds) for which the server will hold this open, while waiting for the client to reestablish the open after a network disconnect.

Open.OperationBuckets[]: An array of 64 elements used to maintain information about outstanding lock and unlock operations performed on resilient **Opens**. An element's zero-based index is referred to as its **BucketIndex**, and its **BucketNumber** is **BucketIndex** + 1. Each element is a structure with the following fields:

- **SequenceNumber:** A rolling 4-bit sequence number which counts from 0 through 15.
- **Free:** A value of FALSE indicates that there is an outstanding lock or unlock request using this **BucketNumber** and **SequenceNumber** combination.

3.2.1.7 Per Pending Request

For each request that was sent to the server and is awaiting a response, the client MUST implement the following:

Request.CancelId: An implementation-dependent identifier generated by the client to support cancellation of pending requests that are sent to the server. The identifier MUST uniquely identify the request among all requests sent by the client to the server.

Request.Message: The contents of the request sent to the server.

Request.Timestamp: The time at which the request was sent to the server.

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 [67](#) choose to implement this timer.

3.2.2.2 Idle Connection Timer

The client SHOULD scan existing connections on a periodic basis, and disconnect connections on which no opens exist and no operations have been issued within an implementation-specific [68](#) time-out.

3.2.3 Initialization

ConnectionTable is set to an empty table.

GlobalFileTable, if implemented, is set to an empty table.

ClientGuid, if implemented, is set to a newly generated GUID.

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 these 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. After sending the request to the server, if the client generates a **CancelId** for the request as specified in section [3.2.4.1.3](#), it is returned to the calling application.

3.2.4.1.1 Signing the Message

If the request message being sent contains a nonzero **SessionId** in the [SMB2 header](#) field, and the session identified by that identifier has **Session.ShouldSign** equal to TRUE, the client MUST sign the request as specified below. If **Session.ShouldSign** is FALSE, the client MAY [<69>](#) sign the request.

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 number of outstanding simultaneous requests that the client can have on a particular connection is determined by the number of credits granted to the client by the server. To maintain its current number of credits, the client MUST set **CreditRequest** to the number of credits that it will consume in sending this request, as specified in sections [3.2.4.1.5](#) and [3.2.4.1.6](#). To increase or decrease this number, the client MUST request the server to grant more or fewer credits than will be consumed by the current request. The client MUST NOT decrease its credits to zero, and SHOULD [<70>](#) request a sufficient number of credits to support implementation-defined local requirements.

Management of credits is initiated by the client and controlled by the server. Specific mechanisms for credit management are implementation defined. [<71>](#)

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](#).

For any message other than [SMB2 CANCEL Request](#) the client MUST take an available identifier from **Connection.SequenceWindow**.

If there is no available identifier, or range of consecutive identifiers for a multi-credit request, as specified in section [3.2.4.1.5](#), the request MUST wait until the necessary identifiers are available before it is sent to the server. The client MAY [<72>](#) send any newly-initiated requests which can be satisfied with available identifiers (including the SMB2 CANCEL Request) to the server on this connection. If the necessary identifiers exceed implementation-defined local requirements, the client MAY fail the request with an implementation-specific error.

When the necessary identifiers are available, the client MUST remove them from **Connection.SequenceWindow**, set **MessageId** in the SMB2 header of the request to the first of these, create a new **Request**, generate a new **CancelId** and assign it to **Request.CancelId**, set **Request.Message** to the SMB2 request being sent to the server, and set **Request.Timestamp** to the current time; and 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.

For an SMB2 CANCEL Request, the client MUST set the **MessageId** field to the identifier that was used for the request that is to be canceled. Because there is no response to the SMB2 CANCEL Request, it MUST NOT be inserted into **Connection.OutstandingRequests**, and the Request Expiration Timer need not be set.

3.2.4.1.4 Sending Compounded Requests

A nonzero 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 and, in such a case, the server SHOULD [fail](#) the requests with STATUS_INVALID_PARAMETER. Compounded requests MUST be aligned on 8-byte boundaries; the last request of the compounded requests does not need to be padded to an 8-byte boundary. If a client or server receives a message that is not aligned on such a boundary, the machine SHOULD [disconnect](#) the connection.

Compounding Unrelated Requests

SMB2_FLAGS_RELATED_OPERATIONS MUST NOT be set in the **Flags** field 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. [fail](#)

Compounding Related Requests

SMB2_FLAGS_RELATED_OPERATIONS MUST be set in the **Flags** field of SMB2 headers on all requests except the first one. The client 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. [fail](#)

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

1. The client MUST construct the initial request as it would if sending the requests separately.
2. It MUST set the **NextCommand** field 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 SMB2_FLAGS_RELATED_OPERATIONS in the **Flags** field of the SMB2 header for this request.
3. The client MUST construct the subsequent request as it would do normally. For any subsequent requests the client MUST set SMB2_FLAGS_RELATED_OPERATIONS in the **Flags** field of the SMB2 header to indicate that it is using the **SessionId**, **TreeId**, and **FileId** supplied in the previous request (or generated by the server in processing that request). The client SHOULD set the **SessionId**, **TreeId**, and **FileId** fields of the request to 0xFFFFFFFFFFFFFFFF, 0xFFFFFFFF, and { 0xFFFFFFFFFFFFFFFF, 0xFFFFFFFFFFFFFFFF }.

3.2.4.1.5 Sending Multi-Credit Requests

A client that implements the SMB 2.1 dialect MAY [send](#) multi-credit requests if **Connection.SupportsMultiCredit** is TRUE. The **CreditCharge** field in the SMB2 header MUST be based on the payload size (the size of the data within the variable-length field) of the request or the anticipated payload size of the response.

$$\text{CreditCharge} \geq (\max(\text{SendPayloadSize}, \text{Expected ResponsePayloadSize}) - 1) / 65536 + 1$$

If the client implements the SMB 2.1 dialect and **Connection.SupportsMultiCredit** is FALSE, **CreditCharge** SHOULD [be](#) set to 0 and the payload size of a request or the anticipated response MUST be a maximum of 64 kilobytes.

If the client does not implement the SMB 2.1 dialect, the **CreditCharge** field MUST be set to 0 and the payload size of a request or the anticipated response MUST be a maximum of 64 kilobytes.

Before sending a multi-credit request, the client MUST consume the calculated number of consecutive **MessageIds** from **Connection.SequenceWindow**.

3.2.4.1.6 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 MUST use the lowest available sequence number in its allowable set for each request.
- For a multi-credit request as specified in section [3.2.4.1.5](#), the client MUST use the lowest available range of consecutive sequence numbers.
- If an [SMB2 CANCEL Request](#) is sent, the client MUST NOT consume a sequence number. Otherwise, the client MUST consume a sequence number, or range of consecutive sequence numbers, when it sends out an SMB2 request.

For the server side of this algorithm, see section [3.3.1.1](#).

3.2.4.2 Application Requests a Connection to a Share

The application provides the following:

- **ServerName**: The name of the server to connect to.
- **ShareName**: The name of the share to connect to.
- **UserCredentials**: An opaque implementation-specific entity that identifies the credentials to be used when authenticating to the remote server.
- **TransportIdentifier**: An optional implementation-specific identifier for the transport on which the connection is to be established.

Upon successful completion, the client MUST return an existing or newly constructed **Session** handle (section [3.2.1.3](#)), an existing or newly constructed **TreeConnect** handle, and the share type (section [3.2.1.4](#)) to the caller.

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 client MUST follow the steps described in the following flow chart. [<79>](#) 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.

The client SHOULD search the **ConnectionTable** and attempt to find an SMB2 connection where **Connection.ServerName** matches the application-supplied **ServerName**. If a connection is found, the client SHOULD use the existing connection. For each existing connection to the target server, the client MUST search through **Connection.SessionTable** for a **Session** that satisfies the client implementation requirements for session reuse. [<80><81>](#)

- If **UserCredentials**, the credentials to be used for the application request, do not match **Session.UserCredentials**, those used in establishing the existing session, the session MUST NOT be reused.
- For operations on an existing Open, the client MUST select the same session that was used to establish the Open.
- The client SHOULD attempt to minimize redundant sessions to the same server.
- The client MAY establish multiple sessions to the same server by the same security context.
- If a new session is being established, the client MAY reuse an existing connection such that multiple sessions are multiplexed on the same connection. If not reusing an existing connection, the client can establish a new connection for the new session.
- The client MUST synchronize simultaneous application requests, as needed, if an existing compatible session is currently being established.

If a matching session is found, the client MUST search through **Session.TreeConnectTable** to find a matching tree connection to the share. If a tree connection is found, the client MUST use the existing tree connection, and no additional steps are required to be performed. If a matching tree connection is not found, the client MUST proceed with establishing a tree connection to the share as described in section [3.2.4.2.4](#).

If a matching session is not found, but a compatible connection is found, the client MAY [<82>](#) either attempt to establish the session on the existing connection, or establish a new connection. If the client is reusing an existing connection, the client MUST perform the following steps:

1. Authenticate to the server as described in section [3.2.4.2.3](#).
2. Establish a new tree connection to the target share as described in section [3.2.4.2.4](#).

Otherwise, the client MUST perform the following steps:

1. Establish a new connection as described in section [3.2.4.2.1](#).
2. Negotiate the protocol as described in section [3.2.4.2.2](#).
3. Authenticate to the server as described in section [3.2.4.2.3](#).
4. Establish a new tree connection to the target share as described in section [3.2.4.2.4](#).

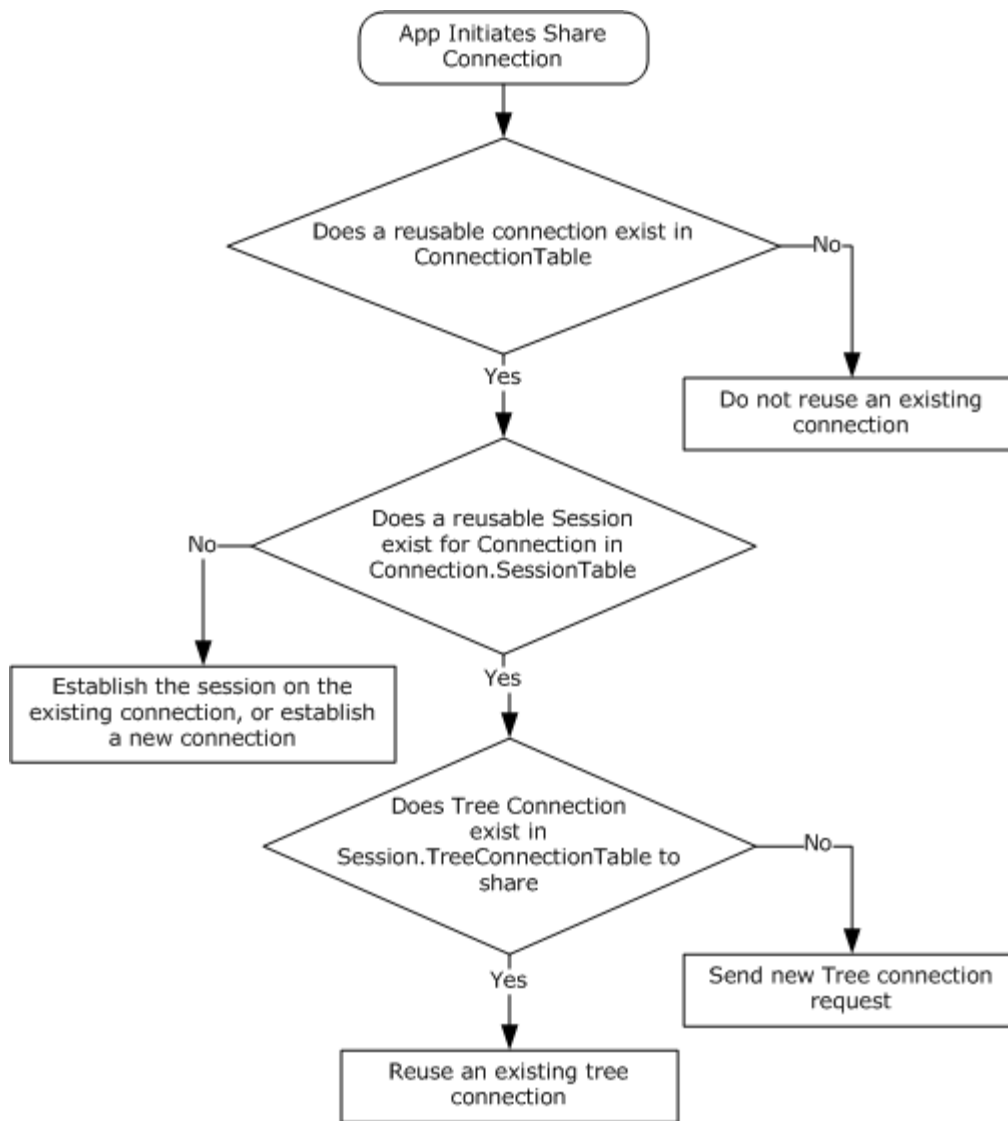


Figure 3: The client MUST follow the steps outlined in this chart.

3.2.4.2.1 Connecting to the Target Server

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. The **ServerName** and the optional **TransportIdentifier** provided by the caller are used to establish the connection. The client SHOULD resolve the **ServerName** as described in [MS-WSQ] section 3.1.1.7, and SHOULD attempt connections to one or more of the returned addresses. The client MAY attempt to initiate each such SMB2 connection on all configured transports that it supports, most commonly Direct TCP and the other transports described in section 2.1. The client MAY choose to prioritize the addresses and/or transport order and try each one sequentially, or try to connect on them all and select one using any implementation-specific heuristic<83>. The client MAY accept the **TransportIdentifier** parameter from the calling application, which specifies what transport to use, and then attempt to use the transport specified. If the connection attempt is successful, a connection object MUST be created, as specified in section 3.2.1.2, 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 [3.2.4.1.6](#), 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.
- **Connection.Dialect**, if implemented, MUST be set to "Unknown".
- **Connection.RequireSigning** MUST be set to **FALSE**.
- **Connection.ServerName** MUST be set to the application-supplied server name.

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 [<84>](#) use either of two possible methods for negotiation.

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 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 Protocol.

3.2.4.2.2.1 Multi-Protocol Negotiate

To negotiate either the SMB 2 Protocol or the SMB Protocol, 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 [2.2.4.5.1](#) and [3.2.4.2](#) and in [\[MS-CIFS\]](#) sections [2.2.4.52](#) and [3.2.4.2.2](#).

If a client does not implement the SMB 2.1 dialect, it MUST perform the following:

- The client MUST include the dialect string "SMB 2.002" [<85>](#) in the list of dialects, along with any other 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.

If a client implements the SMB 2.1 dialect, it MUST perform the following:

- The client MUST include the dialect string "SMB 2.002" and "SMB 2.???" 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 **DialectCount** to 0.
- If the client implements the SMB 2.002 dialect, it MUST do the following:
 - Increment the **DialectCount** by 1.
 - Set the value in **Dialects[DialectCount-1]** array to 0x0202.
- If the client implements the SMB 2.1 dialect, it MUST do the following:
 - Increment the **DialectCount** by 1.
 - Set the value in **Dialects[DialectCount-1]** array to 0x0210.
- If [RequireMessageSigning](#) is TRUE, the client MUST set the SMB2_NEGOTIATE_SIGNING_REQUIRED bit to TRUE in **SecurityMode**. If [RequireMessageSigning](#) is FALSE, the client MUST set the SMB2_NEGOTIATE_SIGNING_ENABLED bit to TRUE in **SecurityMode**.
- Set **Capabilities** and **ClientStartTime** to 0.
- If the client implements the SMB 2.1 dialect, **ClientGuid** MUST be set to the global **ClientGuid** value; otherwise it MUST be set to 0.

This request MUST be sent to the server.

3.2.4.2.3 Authenticating the User

To establish a new session, the client MAY [<86>](#) either:

- 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 3.3.5.2.

OR

- Choose to ignore the **Connection.GSSNegotiateToken** that is received from the server, and initiate a normal GSS sequence, as specified in [\[RFC4178\]](#) 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 3.3.1. The GSS-API output token is up to a size limit determined by local policy [<87>](#) 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.
- The **MessageId** field is set as specified in section [3.2.4.1.3](#).

The SMB2 SESSION_SETUP Request MUST be initialized as follows:

- If [RequireMessageSigning](#) is TRUE, the client MUST set the SMB2_NEGOTIATE_SIGNING_REQUIRED bit in the **SecurityMode** field.

If RequireMessageSigning is FALSE, the client MUST set the SMB2_NEGOTIATE_SIGNING_ENABLED bit in the **SecurityMode** field.
- 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 [<88>](#) either:

- 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 3.3.5.2.
- 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 3.3.1. The GSS-API output token is up to a size limit determined by local policy [<89>](#) 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.
- The **MessageId** field is set as specified in section [3.2.4.1.3](#).
- 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:

- If [RequireMessageSigning](#) is TRUE, the client MUST set the SMB2_NEGOTIATE_SIGNING_REQUIRED bit in the **SecurityMode** field.

If RequireMessageSigning is FALSE, the client MUST set the SMB2_NEGOTIATE_SIGNING_ENABLED bit in the **SecurityMode** field.
- 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

To connect to a share, the client MUST follow the steps outlined below.

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.
- The **MessageId** field is set as specified in section [3.2.4.1.3](#).
- The **SessionId** field is set to **Session.SessionId** of the session that was identified in section [3.2.4.2](#) or established as a result 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. The response from the server MUST be processed as described in section [3.2.5.5](#).

3.2.4.3 Application Requests Opening a File

To open a file on a remote share, the application provides the following:

- A handle to the **TreeConnect** representing the share in which the file to be opened exists.
- The path name of the file being opened, as a DFS pathname for a DFS share, or relative to the **TreeConnect** for a non-DFS share.
- A handle to the **Session** representing the security context of the user opening the file.
- The required 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 or lease state 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 verify the **TreeConnect** and **Session** handles. If the handles are invalid, or if no **TreeConnect** referenced by the tree connect handle is found, or if no **Session** referenced by the session handle is found, the client MUST return an implementation-specific error code locally to the calling application.

If the handles are valid and a **TreeConnect** and **Session** are found, the caller MUST ensure that the supplied **TreeConnect** is valid within the **Session**. **TreeConnect.Session** MUST match the **Session**.

The client MUST use the **Connection** referenced by **Session.Connection** to send the request to the server.

If the client supports leasing and the server supports leasing as identified by **Connection.SupportsLeasing**, the client MUST search the **GlobalFileTable** for an entry matching one of the following:

- The application-supplied PathName if **TreeConnect.IsDfsShare** is TRUE.
- The concatenation of **Connection.ServerName**, **TreeConnect.ShareName**, and the application-supplied PathName, joined with pathname separators (example: server\share\path), if **TreeConnect.IsDfsShare** is FALSE.

If an entry is not found, a new File entry MUST be created and added to the **GlobalFileTable** and a **File.LeaseKey**, as specified in section [3.2.1.5](#), MUST be assigned to the entry. **File.OpenTable** MUST be initialized to an empty table and **File.LeaseState** MUST be initialized to SMB2_LEASE_NONE.

If the client accesses a file through multiple paths, such as using different server names or share names or parent directory names, it will create multiple **File** elements, and therefore multiple **File.LeaseKeys** for the same remote file. This loses the performance benefits of sharing cache state across all **Opens** of the same file, and may cause additional lease breaks to be generated, as actions by a client through one path will affect caching by that client through other paths. However, the impact is a matter of performance; cache correctness is preserved.

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.
- The **MessageId** field is set as specified in section [3.2.4.1.3](#).
- The **SessionId** field is set to **TreeConnect.Session.SessionId**.
- The **TreeId** field is set to **TreeConnect.TreeConnectId**.
- If **TreeConnect.IsDfsShare** is TRUE, the SMB2_FLAGS_DFS_OPERATIONS flag is set in the **Flags** field.

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 implementation-specific oplock level. [<90>](#)
- 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 application-supplied path into the **Buffer**, and sets the **NameLength** to the length, in bytes, of the path and the **NameOffset** to the offset, in bytes, to the 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. The response from the server MUST be processed as described in section [3.2.5.7](#).

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 that **TreeConnect.ShareName** will 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 **TreeConnect.ShareName** will 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 provides 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 with this CREATE request.

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 provides 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 with this CREATE request.

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 provides 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 appends it to any other create contexts being issued with this CREATE request. 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 provides 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 with this CREATE request.

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 provides 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 appends it to any other create contexts being issued with this CREATE request.

3.2.4.3.8 Requesting a Lease on a File

To request a lease, in addition to the parameters that are specified in [3.2.4.3](#), the application provides a Boolean value indicating that a lease requires to be taken and a LeaseState value (as defined in section [2.2.13.2.8](#)) that indicates the type of lease to be requested. [<91>](#)

If the client does not support the SMB 2.1 dialect, the client MUST fail this request with STATUS_NOT_SUPPORTED.

If **Connection.Dialect** is not equal to "2.100" or **Connection.SupportsLeasing** is FALSE, the client MUST fail this request with STATUS_NOT_SUPPORTED.

The client MUST construct an **SMB2_CREATE request** as described in [3.2.4.3](#), with a RequestedOplockLevel of SMB2_OPLOCK_LEVEL_LEASE. The client MUST attach an SMB2_CREATE_REQUEST_LEASE create context to the request. The create context MUST be formatted as described in section [2.2.13.2.8](#), with the **LeaseState** value provided by the application.

3.2.4.3.9 Application Requests Maximal Access Information of a File

To request maximal access information of a file being opened or created, in addition to the parameters that are specified in section [3.2.4.3](#), the application provides a Boolean indicating

whether it is requesting maximal access information of a file, and optionally a Timestamp value, in [FILETIME](#) format as specified in [\[MS-DTYP\]](#) section 2.3.1. If the application is requesting this information, the client MUST construct an [SMB2_CREATE_QUERY_MAXIMAL_ACCESS_REQUEST](#) create context using the syntax specified in section [2.2.13.2.5](#). The client appends it to any other create contexts being issued with this CREATE request.

3.2.4.3.10 Application Requests Identifier of a File

To request an identifier of a file being opened or created, the client MUST construct an [SMB2_CREATE_QUERY_ON_DISK_ID](#) create context using the syntax specified in section [2.2.13.2](#). The client appends it to any other create contexts being issued with this CREATE request.

3.2.4.4 Re-establishing a Durable Open

When an application requests an operation on a durable open that existed on a now-disconnected connection, that is, **Open.Connection** is NULL, and **Open.Durable** is TRUE, then the client SHOULD attempt to reconnect to this open as specified here.

The client MUST attempt to connect to the target share, as specified in section [3.2.4.2](#), by obtaining the name of the server and the name of the share to connect to from the **Open.FileName**. 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.
- The **MessageId** field is set as specified in section [3.2.4.1.3](#).
- The **SessionId** field is set to **TreeConnect.Session.SessionId**.
- The **TreeId** field is set to **TreeConnect.TreeConnectId**.

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.

- If a client implements the SMB 2.1 dialect, and the original open was performed using a lease as described in section [3.2.4.3.8](#), it MUST also implement the following:
 - The client MUST re-request the lease as described in section [3.2.4.3.8](#).

This request MUST be sent to the server.

3.2.4.5 Application Requests Closing a File or Named Pipe

The application provides:

- A handle to the **Open** identifying the file or named pipe to be closed.
- A Boolean that, if set, specifies that it requires the attributes of the file after the close is executed.

If the handle is invalid, or if no **Open** referenced by the handle is found, the client MUST return an implementation-specific error code. If the handle is valid, and **Open** is found, the client MUST proceed as follows.

If **Open.Connection** is NULL, and **Open.Durable** is TRUE, the client SHOULD attempt to reconnect to this open, as specified in section [3.2.4.4](#). If the reconnect succeeds, the close MUST be retried. If it fails, the error code MUST be returned to the application.

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

If **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.
- The **MessageId** field is set as specified in section [3.2.4.1.3](#).
- The **SessionId** field is set to **Open.TreeConnect.Session.SessionId**.
- The **TreeId** field is set to **Open.TreeConnect.TreeConnectId**.

The SMB2 CLOSE Request MUST be initialized as follows:

- If the application requires 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.

3.2.4.6 Application Requests Reading from a File or Named Pipe

The application provides:

- A handle to the **Open** identifying 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).

If the handle is invalid, or if no **Open** referenced by the handle is found, the client MUST return an implementation-specific error code. If the handle is valid and **Open** is found, the client MUST proceed as follows.

If **Open.Connection** is NULL, and **Open.Durable** is TRUE, the client SHOULD attempt to reconnect to this **Open**, as specified in section [3.2.4.4](#). If the reconnect succeeds, the read MUST be retried. If it fails, the error code MUST be returned to the application.

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

If **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.
- The **SessionId** field is set to **Open.TreeConnect.Session.SessionId**.
- The **TreeId** field is set to **Open.TreeConnect.TreeConnectId**.

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. [<92>](#)

If a client implements the SMB 2.1 dialect requests reading from a file and if **Connection.SupportsMultiCredit** is TRUE, the **CreditCharge** field in the SMB2 header MUST be set to $(1 + (\text{Length} - 1) / 65536)$.

The **MessageId** field in the SMB2 header is set as specified in section [3.2.4.1.3](#), and the request is sent to the server.

3.2.4.7 Application Requests Writing to a File or Named Pipe

The application provides:

- A handle to the **Open** identifying 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.

If the handle is invalid, or if no **Open** referenced by the handle is found, the client MUST return an implementation-specific error code. If the handle is valid and **Open** is found, the client MUST proceed as follows.

If **Open.Connection** is NULL, and **Open.Durable** is TRUE, the client SHOULD attempt to reconnect to this open as specified in section [3.2.4.4](#). If the reconnect succeeds, the write MUST be retried. If it fails, the error code MUST be returned to the application.

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

If **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.
- The **SessionId** field is set to **Open.TreeConnect.Session.SessionId**.
- The **TreeId** field is set to **Open.TreeConnect.TreeConnectId**.

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 fill the bytes, if any, between the beginning of the **Buffer** field and the beginning of the data (at **DataOffset**) with zeros. [<93>](#)

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 [<94>](#) send these separate writes in any order.

If a client that implements the SMB 2.1 dialect requests writing to a file and if **Connection.SupportsMultiCredit** is TRUE, the **CreditCharge** field in the SMB2 header MUST be set to $(1 + (\text{Length} - 1) / 65536)$.

The **MessageId** field in the SMB2 header is set as specified in section [3.2.4.1.3](#), and the request is sent to the server.

3.2.4.8 Application Requests Querying File Attributes

The application provides:

- A handle to the **Open** identifying 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 MUST provide the following:
 - A Boolean indicating whether to restart the EA scan.
 - A Boolean indicating whether only a single entry must be returned.

It also MAY provide one of the following:

- The index of the first EA entry to return from the array of extended attributes that are associated with the file or named pipe. An index value of 1 corresponds to the first extended attribute.
- A list of FILE_GET_EA_INFORMATION structures, as specified in [\[MS-FSCC\]](#) section 2.4.15.1.

If the handle is invalid, or if no **Open** referenced by the handle is found, the client MUST return an implementation-specific error code. If the handle is valid and **Open** is found, the client MUST proceed as follows.

If **Open.Connection** is NULL and **Open.Durable** is **TRUE**, the client SHOULD attempt to reconnect to this open, as specified in section [3.2.4.4](#). If the reconnect succeeds, the query MUST be retried. If it fails, the error code MUST be returned to the application.

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

If **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.
- The **MessageId** field is set as specified in section [3.2.4.1.3](#).
- The **SessionId** field is set to **Open.TreeConnect.Session.SessionId**.
- The **TreeId** field is set to **Open.TreeConnect.TreeConnectId**.

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. An **OutputBufferLength** exceeding **Connection.MaxTransactSize** will be rejected by the server.
- If the query is for FileFullEaInformation and the application has provided a list of EAs to query, the **InputBufferOffset** field MUST be set to the offset of the **Buffer** field from the start of the SMB2 header. Otherwise, the **InputBufferOffset** field SHOULD be set to 0. [<95>](#)
- If the query is for **FileFullEaInformation** and the application has provided a list of EAs to query, the **InputBufferLength** field MUST be set to the length of the FILE_GET_EA_INFORMATION buffer provided by the application, as specified in [\[MS-FSCC\]](#) section 2.4.15.1. Otherwise, the **InputBufferLength** field MUST be set to 0.
- If the query is for **FileFullEaInformation**, and the application has not provided a list of EAs to query, but has provided an extended attribute index, the **AdditionalInformation** field MUST be

set to the extended attribute index provided by the calling application. Otherwise, the **AdditionalInformation** field MUST be set to 0.

- If the query is for **FileFullEaInformation**, the **Flags** field in the SMB2 QUERY_INFO request MUST be set to zero or more of the following bit flags. Otherwise, it MUST be set to 0.
 - SL_RESTART_SCAN if the application requested that the EA scan be restarted.
 - SL_RETURN_SINGLE_ENTRY if the application requested that only a single entry be returned.
 - SL_INDEX_SPECIFIED if the application provided an EA index instead of a list of EAs.
- The **FileId** field is set to **Open.FileId**.

The request MUST be sent to the server.

3.2.4.9 Application Requests Applying File Attributes

The application provides:

- A handle to the **Open** identifying 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.

If the handle is invalid, or if no **Open** referenced by the handle is found, the client MUST return an implementation-specific error code. If the handle is valid and **Open** is found, the client MUST proceed as follows.

If **Open.Connection** is NULL, and **Open.Durable** is TRUE, the client SHOULD attempt to reconnect to this open, as specified in section [3.2.4.4](#). If the reconnect succeeds, the set MUST be retried. If it fails, the error code MUST be returned to the application.

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

If **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.
- The **MessageId** field is set as specified in section [3.2.4.1.3](#).
- The **SessionId** field is set to **Open.TreeConnect.Session.SessionId**.
- The **TreeId** field is set to **Open.TreeConnect.TreeConnectId**.

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. A **BufferLength** exceeding **Connection.MaxTransactSize** will be rejected by the server.
- 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 Querying File System Attributes

The application provides:

- A handle to the **Open** identifying a file that resides in 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.

If the handle is invalid, or if no **Open** referenced by the handle is found, the client MUST return an implementation-specific error code. If the handle is valid and **Open** is found, the client MUST proceed as follows.

If **Open.Connection** is NULL, and **Open.Durable** is **TRUE**, the client SHOULD attempt to reconnect to this open, as specified in section [3.2.4.4](#). If the reconnect succeeds, the query MUST be retried. If it fails, the error code MUST be returned to the application.

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

If **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.
- The **MessageId** field is set as specified in section [3.2.4.1.3](#).
- The **SessionId** field is set to **Open.TreeConnect.Session.SessionId**.
- The **TreeId** field is set to **Open.TreeConnect.TreeConnectId**.

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. An **OutputBufferLength** exceeding **Connection.MaxTransactSize** will be rejected by the server.
- The **InputBufferOffset** field SHOULD [<96>](#) be 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.

3.2.4.11 Application Requests Applying File System Attributes

The application provides:

- A handle to the **Open** identifying a file that resides in 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.

If the handle is invalid, or if no **Open** referenced by the handle is found, the client MUST return an implementation-specific error code. If the handle is valid and **Open** is found, the client MUST proceed as follows.

If **Open.Connection** is NULL, and **Open.Durable** is TRUE, the client SHOULD attempt to reconnect to this open, as specified in section [3.2.4.4](#). If the reconnect succeeds, the set MUST be retried. If it fails, the error code MUST be returned to the application.

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

If **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.
- The **MessageId** field is set as specified in section [3.2.4.1.3](#).
- The **SessionId** field is set to **Open.TreeConnect.Session.SessionId**.
- The **TreeId** field is set to **Open.TreeConnect.TreeConnectId**.

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. A **BufferLength** exceeding **Connection.MaxTransactSize** will be rejected by the server.
- The **AdditionalInformation** is set to 0.
- The **FileId** field is set to **Open.FileId**.

The request MUST be sent to the server.

3.2.4.12 Application Requests Querying File Security

The application provides:

- A handle to the **Open** identifying 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](#).

If the handle is invalid, or if no **Open** referenced by the handle is found, the client MUST return an implementation-specific error code. If the handle is valid and **Open** is found, the client MUST proceed as follows.

If **Open.Connection** is NULL, and **Open.Durable** is TRUE, the client SHOULD attempt to reconnect to this open, as specified in section [3.2.4.4](#). If the reconnect succeeds, the query MUST be retried. If it fails, the error code MUST be returned to the application.

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

If **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.
- The **MessageId** field is set as specified in section [3.2.4.1.3](#).
- The **SessionId** field is set to **Open.TreeConnect.Session.SessionId**.
- The **TreeId** field is set to **Open.TreeConnect.TreeConnectId**.

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. An **OutputBufferLength** exceeding **Connection.MaxTransactSize** will be rejected by the server.
- The **InputBufferOffset** field SHOULD [<97>](#) be 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.

3.2.4.13 Application Requests Applying File Security

The application provides:

- A handle to the **Open** identifying 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 requires to set for the file, as specified in section [2.2.37](#).

If the handle is invalid, or if no **Open** referenced by the handle is found, the client MUST return an implementation-specific error code. If the handle is valid and **Open** is found, the client MUST proceed as follows.

If **Open.Connection** is NULL, and **Open.Durable** is **TRUE**, the client SHOULD attempt to reconnect to this open, as specified in section [3.2.4.4](#). If the reconnect succeeds, the set MUST be retried. If it fails, the error code MUST be returned to the application.

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

If **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.
- The **MessageId** field is set as specified in section [3.2.4.1.3](#).
- The **SessionId** field is set to **Open.TreeConnect.Session.SessionId**.
- The **TreeId** field is set to **Open.TreeConnect.TreeConnectId**.

The SMB2 SET_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 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. A **BufferLength** exceeding **Connection.MaxTransactSize** will be rejected by the server.
- 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.

3.2.4.14 Application Requests Querying Quota Information

The application provides:

- A handle to the **Open** identifying 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 a list of the SIDs whose quota information is to be queried, in the form of a SidList of FILE_GET_QUOTA_INFORMATION structures linked via the **NextOffset** field.

If the handle is invalid, or if no **Open** referenced by the handle is found, the client MUST return an implementation-specific error code. If the handle is valid and **Open** is found, the client MUST proceed as follows.

If **Open.Connection** is NULL, and **Open.Durable** is **TRUE**, the client SHOULD attempt to reconnect to this open, as specified in section [3.2.4.4](#). If the reconnect succeeds, the query MUST be retried. If it fails, the error code MUST be returned to the application.

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

If **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.
- The **MessageId** field is set as specified in section [3.2.4.1.3](#).
- The **SessionId** field is set to **Open.TreeConnect.Session.SessionId**.
- The **TreeId** field is set to **Open.TreeConnect.TreeConnectId**.

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. An **OutputBufferLength** exceeding **Connection.MaxTransactSize** will be rejected by the server.
- 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 **SidBuffer** field of the SMB2 QUERY_INFO structure field, and initialized as follows:
 - If only a single entry is to be returned, the client sets **ReturnSingle** to **TRUE**. Otherwise, it is set to FALSE.
 - If the application requires 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 the application provides a SidList, via one or more FILE_GET_QUOTA_INFORMATION structures linked by **NextEntryOffset**, they MUST be copied to the beginning of the **SidBuffer**, **SidListLength** MUST be set to their length in bytes, **StartSidLength** SHOULD be set to 0, and **StartSidOffset** SHOULD be set to 0. [<98>](#)
- If a SidList is not provided by the application, then **SidListLength** MUST be set to 0, **StartSidLength** SHOULD be set to 0, and **StartSidOffset** SHOULD be 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 buffer for the SidList.

The request MUST be sent to the server.

3.2.4.15 Application Requests Applying Quota Information

The application provides:

- A handle to the **Open** identifying a directory.
- A list of SIDs (as specified in [\[MS-DTYP\]](#) section 2.4.2) for which quota information is to be applied.
- For each SID, the quota warning threshold and quota limit to be applied, as specified in [\[MS-FSCCI\]](#) section 2.4.33.

If the handle is invalid, or if no **Open** referenced by the handle is found, the client MUST return an implementation-specific error code. If the handle is valid and **Open** is found, the client MUST proceed as follows.

If **Open.Connection** is NULL, and **Open.Durable** is **TRUE**, the client SHOULD attempt to reconnect to this open, as specified in section [3.2.4.4](#). If the reconnect succeeds, the set MUST be retried. If it fails, the error code MUST be returned to the application.

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

If **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.
- The **MessageId** field is set as specified in section [3.2.4.1.3](#).
- The **SessionId** field is set to **Open.TreeConnect.Session.SessionId**.
- The **TreeId** field is set to **Open.TreeConnect.TreeConnectId**.

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 **AdditionalInformation** is set to 0.
- The **FileId** field is set to **Open.FileId**.

- The **Buffer** field is set to one or more FILE_QUOTA_INFORMATION structures, as specified in [\[MS-FSCC\]](#) section 2.4.33.
 - The **NextEntryOffset** field is set to the offset, in bytes, to the next FILE_QUOTA_INFORMATION structure, or zero if this is the last structure in the buffer.
 - The **Sid** field is set to the application-provided SID, in little-endian binary format as specified in [\[MS-DTYP\]](#) section 2.4.2.2.
 - The **SidLength** field is set to the length of the **Sid** field, in bytes.
 - The **ChangeTime** field is set to the current time, as specified in [\[MS-DTYP\]](#) section 2.3.1.
 - The **QuotaUsed** field is ignored and can be set to any value.
 - The **QuotaThreshold** field is set to the application provided quota warning threshold.
 - The **QuotaLimit** field is set to the application provided quota limit.
- The **BufferLength** is set to the length, in bytes, of the **Buffer** field. A **BufferLength** exceeding **Connection.MaxTransactSize** will be rejected by the server.
- The **BufferOffset** is set to the offset to the **Buffer**, in bytes, from the beginning of the SMB2 header.

The request MUST be sent to the server.

3.2.4.16 Application Requests Flushing Cached Data

The application provides:

- A handle to the **Open** identifying a file or named pipe for which it requires to flush cached data.

If the handle is invalid, or if no **Open** referenced by the handle is found, the client MUST return an implementation-specific error code. If the handle is valid and **Open** is found, the client MUST proceed as follows.

If **Open.Connection** is NULL, and **Open.Durable** is TRUE, the client SHOULD attempt to reconnect to this open, as specified in section [3.2.4.4](#). If the reconnect succeeds, the flush MUST be retried. If it fails, the error code MUST be returned to the application.

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

If **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.
- The **MessageId** field is set as specified in section [3.2.4.1.3](#).
- The **SessionId** field is set to **Open.TreeConnect.Session.SessionId**.
- The **TreeId** field is set to **Open.TreeConnect.TreeConnectId**.

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.

3.2.4.17 Application Requests Enumerating a Directory

The application provides:

- A handle to the **Open** identifying 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.

If the handle is invalid, or if no **Open** referenced by the handle is found, the client MUST return an implementation-specific error code. If the handle is valid and **Open** is found, the client MUST proceed as follows.

If **Open.Connection** is NULL, and **Open.Durable** is TRUE, the client SHOULD attempt to reconnect to this open, as specified in section [3.2.4.4](#). If the reconnect succeeds, the enumeration MUST be retried. If it fails, the error code MUST be returned to the application.

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

If **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.
- The **SessionId** field is set to **Open.TreeConnect.Session.SessionId**.
- The **TreeId** field is set to **Open.TreeConnect.TreeConnectId**.

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. An **OutputBufferLength** exceeding **Connection.MaxTransactSize** will be rejected by the server.
- 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. <99>
- If the application requested that only a single entry be returned, the client sets SMB2_RETURN_SINGLE_ENTRY to TRUE in the **Flags** field.

If the client implements the SMB 2.1 dialect and if **Connection.SupportsMultiCredit** is TRUE, the **CreditCharge** field in the SMB2 header MUST be set to $(1 + (\text{OutputBufferLength} - 1) / 65536)$.

The **MessageId** field in the SMB2 header is set as specified in section [3.2.4.1.3](#), and the request is sent to the server.

3.2.4.17.1 Application Requests Continuing a Directory Enumeration

If an application requires to continue an enumeration for which only partial results were previously returned, it does so by executing a request, as specified in section [3.2.4.17](#), but makes sure it does not request restarting the enumeration. Doing so allows it to continue a previous enumeration.

3.2.4.18 Application Requests Change Notifications for a Directory

The application provides:

- A handle to the **Open** identifying 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 requires to be notified when changes occur and does not require to see the actual changes, the maximum output buffer MUST be set to 0.

If the handle is invalid, or if no **Open** referenced by the handle is found, the client MUST return an implementation-specific error code. If the handle is valid and **Open** is found, the client MUST proceed as follows.

If **Open.Connection** is NULL, and **Open.Durable** is TRUE, the client SHOULD attempt to reconnect to this open, as specified in section [3.2.4.4](#). If the reconnect succeeds, the change notify MUST be retried. If it fails, the error code MUST be returned to the application.

If **Open.Connection** is NULL, and **Open.Durable** is FALSE, the client MUST fail the change notify operation.

If **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.
- The **MessageId** field is set as specified in section [3.2.4.1.3](#).
- The **SessionId** field is set to **Open.TreeConnect.Session.SessionId**.
- The **TreeId** field is set to **Open.TreeConnect.TreeConnectId**.

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. An **OutputBufferLength** exceeding **Connection.MaxTransactSize** will be rejected by the server.
- 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.

3.2.4.19 Application Requests Locking of an Array of Byte Ranges

The application provides:

- A handle to the **Open** identifying 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.

If the handle is invalid, or if no **Open** referenced by the handle is found, the client MUST return an implementation-specific error code. If the handle is valid and **Open** is found, the client MUST proceed as follows.

If **Open.Connection** is NULL, and **Open.Durable** is TRUE, the client SHOULD attempt to reconnect to this open as specified in section [3.2.4.4](#). If the reconnect succeeds, the lock MUST be retried. If it fails, the error code MUST be returned to the application.

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

If **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.
- The **MessageId** field is set as specified in section [3.2.4.1.3](#).

- The **SessionId** field is set to **Open.TreeConnect.Session.SessionId**.
- The **TreeId** field is set to **Open.TreeConnect.TreeConnectId**.

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_ELEMENT](#) 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.

If **Open.ResilientHandle** is TRUE, the client MUST do the following:

- The client MUST scan through **Open.OperationBuckets** and find an element with its **Free** field set to TRUE. If no such element could be found, an implementation-specific error MUST be returned to the application.
- Let the zero-based array index of the element chosen above be referred to as **BucketIndex**, and let **BucketNumber** = **BucketIndex** + 1.
- Set **Open.OperationBuckets[BucketIndex].Free** = FALSE
- Let the **SequenceNumber** of the element chosen above be referred to as **BucketSequence**.
- The **LockSequence** field of the SMB2 lock request MUST be set to (**BucketNumber** << 4) + **BucketSequence**.
- Increment the **SequenceNumber** of the element chosen above using MOD 16 arithmetic.

The request MUST be sent to the server.

3.2.4.20 Application Requests an IO Control Code Operation

3.2.4.20.1 Application Requests Enumeration of Previous Versions

The application provides:

- A handle to the **Open** identifying a file on a volume for which the application requires the previous version time stamps.
- The maximum output buffer size that it will accept.

If the handle is invalid, or if no **Open** referenced by the handle is found, the client MUST return an implementation-specific error code. If the handle is valid and **Open** is found, the client MUST proceed as follows.

If **Open.Connection** is NULL, and **Open.Durable** is TRUE, the client SHOULD attempt to reconnect to this open, as specified in section [3.2.4.4](#). If the reconnect succeeds, this FSCTL MUST be retried. If it fails, the error code MUST be returned to the application.

If **Open.Connection** is NULL, and **Open.Durable** is FALSE, the client MUST fail this FSCTL operation.

If **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.
- The **MessageId** field is set as specified in section [3.2.4.1.3](#).
- The **SessionId** field is set to **Open.TreeConnect.Session.SessionId**.
- The **TreeId** field is set to **Open.TreeConnect.TreeConnectId**.

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 SHOULD [<100>](#) be set to 0.
- The **InputCount** field is set to 0.
- The **OutputOffset** field SHOULD [<101>](#) be set to zero.
- 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.

3.2.4.20.2 Application Requests a Server-Side Data Copy

Requesting a server-side data copy occurs in several steps. These are outlined in the following diagram:

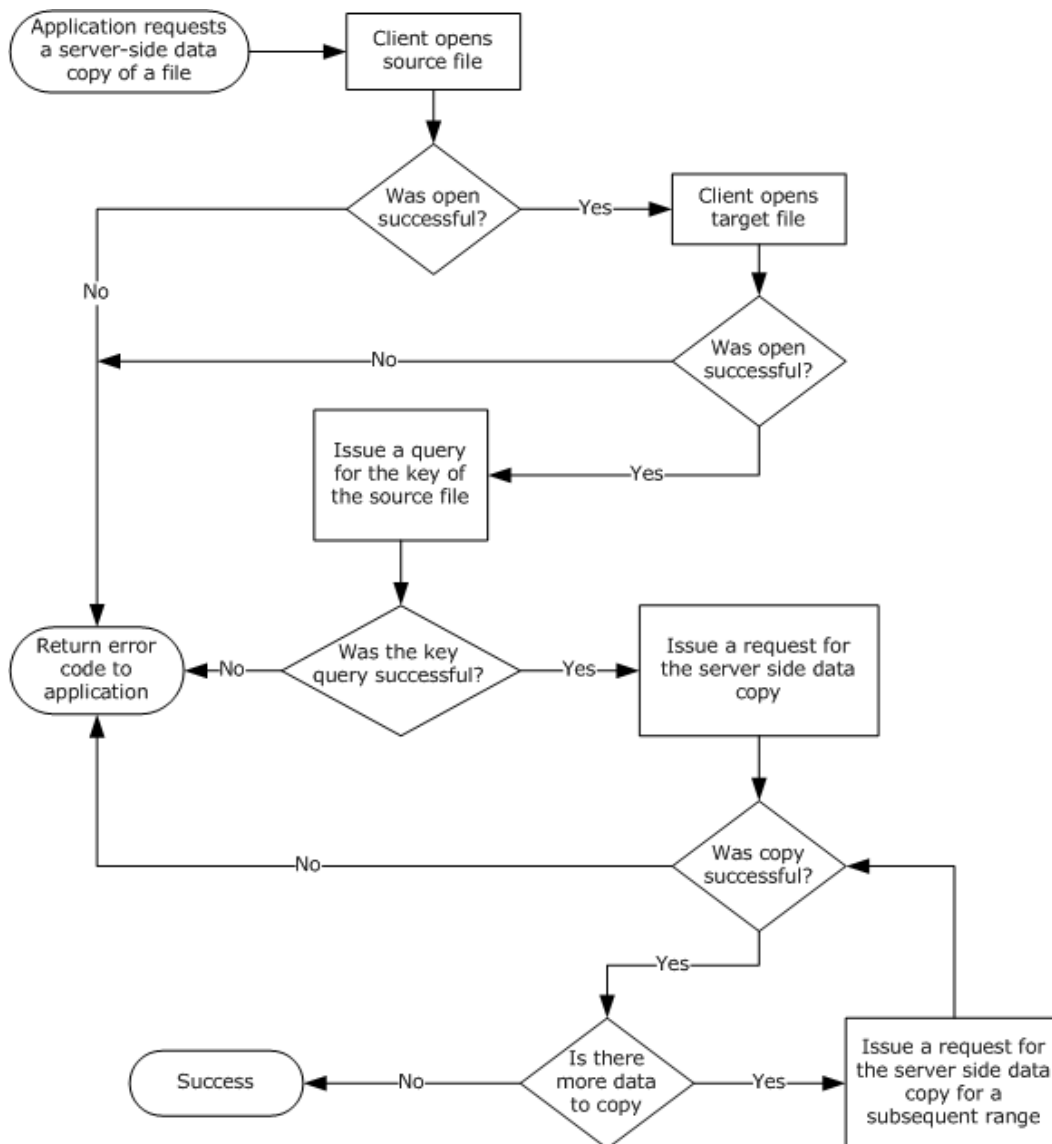


Figure 4: Application requesting a server-side data copy

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.

3.2.4.20.2.1 Application Requests a Source File Key

The application provides:

- A handle to the Open identifying a file for which the application requires a key to use in server-side data operations.

If the handle is invalid, or if no **Open** referenced by the handle is found, the client **MUST** return an implementation-specific error code. If the handle is valid and **Open** is found, the client **MUST** proceed as follows.

If **Open.Connection** is NULL, and **Open.Durable** is TRUE, the client SHOULD attempt to reconnect to this open, as specified in section [3.2.4.4](#). If the reconnect succeeds, this FSCTL MUST be retried. If it fails, the error code MUST be returned to the application.

If **Open.Connection** is NULL, and **Open.Durable** is FALSE, the client MUST fail this FSCTL operation.

If **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.
- The **MessageId** field is set as specified in section [3.2.4.1.3](#).
- The **SessionId** field is set to **Open.TreeConnect.Session.SessionId**.
- The **TreeId** field is set to **Open.TreeConnect.TreeConnectId**.

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 SHOULD [<102>](#) be set to 0.
- The **InputCount** field is set to 0.
- The **OutputOffset** field SHOULD [<103>](#) be set to 0.
- 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.

3.2.4.20.2.2 Application Requests a Server Side Data Copy

The application provides:

- A handle to the Open identifying the destination file.
- The FSCTL code for the server side copy, either FSCTL_SRV_COPYCHUNK or FSCTL_SRV_COPYCHUNK_WRITE. [<104>](#)
- 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.

If the handle is invalid, or if no **Open** referenced by the handle is found, the client MUST return an implementation-specific error code. If the handle is valid and **Open** is found, the client MUST proceed as follows.

If **Open.Connection** is NULL, and **Open.Durable** is TRUE, the client SHOULD attempt to reconnect to this open, as specified in section [3.2.4.4](#). If the reconnect succeeds, this FSCTL MUST be retried. If it fails, the error code MUST be returned to the application.

If **Open.Connection** is NULL, and **Open.Durable** is FALSE, the client MUST fail this FSCTL operation.

If **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.
- The **MessageId** field is set as specified in section [3.2.4.1.3](#).
- The **SessionId** field is set to **Open.TreeConnect.Session.SessionId**.
- The **TreeId** field is set to **Open.TreeConnect.TreeConnectId**.

The SMB2 IOCTL Request MUST be initialized as follows:

- The **CtlCode** field MUST be set to the FSCTL code supplied by 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** 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 [<105>](#) be set to zero.
- 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.

3.2.4.20.3 Application Requests DFS Referral Information

The application provides the following:

- **ServerName**: The name of the server from which to query referrals.
- **UserCredentials**: An opaque implementation-specific entity that identifies the credentials to be used when authenticating to the remote server.
- 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 search for an existing **Session** and **TreeConnect** to any share on the server identified by **ServerName** for the user identified by **UserCredentials**. If no **Session** and **TreeConnect** are found, the client MUST establish a new **Session** and **TreeConnect** to IPC\$ on the target server as described in section [3.2.4.2](#) using the supplied **ServerName** and **UserCredentials**.

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.
- The **MessageId** field is set as specified in section [3.2.4.1.3](#).
- The **SessionId** field is set to **Session.SessionId**.
- The **TreeId** field is set to **TreeConnect.TreeConnectId**.

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 **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 **RequestFileName** is set to the path for which the referral information is being queried.
- The **InputCount** field is set to the size of the constructed **REQ_GET_DFS_REFERRAL** structure.
- The **OutputOffset** field SHOULD [<106>](#) be set to zero.
- 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 using the **Session** and **TreeConnect** obtained as a result of connecting to the IPC\$ share on the server.

3.2.4.20.4 Application Requests a Pipe Transaction

The application provides:

- A handle to the **Open** identifying the named pipe on which to issue the operation.
- An input buffer.
- A maximum output buffer response size, in bytes.

If the handle is invalid, or if no **Open** referenced by the handle is found, the client MUST return an implementation-specific error code. If the handle is valid and **Open** is found, the client MUST proceed as follows.

If **Open.Connection** is NULL, and **Open.Durable** is TRUE, the client SHOULD attempt to reconnect to this open, as specified in section 3.2.4.4. If the reconnect succeeds, this FSCTL MUST be retried. If it fails, the error code MUST be returned to the application.

If **Open.Connection** is NULL, and **Open.Durable** is FALSE, the client MUST fail this FSCTL operation.

If **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.
- The **MessageId** field is set as specified in section 3.2.4.1.3.
- The **SessionId** field is set to **Open.TreeConnect.Session.SessionId**.
- The **TreeId** field is set to **Open.TreeConnect.TreeConnectId**.

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 [<107>](#) be set to zero.
- 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.

3.2.4.20.5 Application Requests a Peek at Pipe Data

The application provides:

- A handle to the **Open** identifying the named pipe on which to issue the operation.
- The number of bytes to peek at in the pipe buffer.

If the handle is invalid, or if no **Open** referenced by the handle is found, the client MUST return an implementation-specific error code. If the handle is valid and **Open** is found, the client MUST proceed as follows.

If **Open.Connection** is NULL, and **Open.Durable** is TRUE, the client SHOULD attempt to reconnect to this open, as specified in section [3.2.4.4](#). If the reconnect succeeds, this FSCTL MUST be retried. If it fails, the error code MUST be returned to the application.

If **Open.Connection** is NULL, and **Open.Durable** is FALSE, the client MUST fail this FSCTL operation.

If **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.
- The **MessageId** field is set as specified in section [3.2.4.1.3](#).
- The **SessionId** field is set to **Open.TreeConnect.Session.SessionId**.
- The **TreeId** field is set to **Open.TreeConnect.TreeConnectId**.

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 SHOULD [<108>](#) be set to 0.
- The **InputCount** field is set to 0.
- The **OutputOffset** field SHOULD [<109>](#) be set to zero.
- 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 requires to peek at.
- SMB2_0_IOCTL_IS_FSCTL is set to TRUE in the **Flags** field.

The request MUST be sent to the server.

3.2.4.20.6 Application Requests a Pass-Through Operation

An SMB2 server MAY [<110>](#) support pass-through operation requests.

The application provides:

- A handle to the **Open** identifying 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.

If the handle is invalid, or if no **Open** referenced by the handle is found, the client MUST return an implementation-specific error code. If the handle is valid and **Open** is found, the client MUST proceed as follows.

If **Open.Connection** is NULL, and **Open.Durable** is TRUE, the client SHOULD attempt to reconnect to this open, as specified in section [3.2.4.4](#). If the reconnect succeeds, the FSCTL or IOCTL MUST be retried. If it fails, the error code MUST be returned to the application.

If **Open.Connection** is NULL, and **Open.Durable** is FALSE, the client MUST fail the FSCTL or IOCTL operation.

If **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.
- The **MessageId** field is set as specified in section [3.2.4.1.3](#).
- The **SessionId** field is set to **Open.TreeConnect.Session.SessionId**.
- The **TreeId** field is set to **Open.TreeConnect.TreeConnectId**.

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 [<111>](#) be set to zero.
- The **OutputCount** field is set to 0.

- The **MaxInputResponse** field is set to the maximum input buffer response size, in bytes, that the application will accept.
- The **MaxOutputResponse** field is set to the maximum output buffer response 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.

3.2.4.20.7 Application Requests Content Information for a File

An application can request Content Information from the server that contains a set of hashes that can be used by the application to retrieve the contents of a specific file using the branch cache, as specified in [\[MS-PCCRC\]](#). This request is only valid for the SMB 2.1 dialect. To retrieve the Content Information, the application provides the following:

- The **HashType**, as specified in section [2.2.31.2](#).
- The **HashVersion**, as specified in section [2.2.31.2](#).
- The **HashRetrievalType**, as specified in section [2.2.31.2](#).
- A handle to the **Open** identifying the remote file with which the Content Information is associated.
- The maximum number of bytes to get from the associated Content Information data structure.
- The offset into the Content Information data structure, if the structure is being retrieved across multiple requests, in bytes.

If the handle is invalid, or if no **Open** referenced by the handle is found, the client MUST return an implementation-specific error code. If the handle is valid and **Open** is found, the client MUST proceed as follows.

If **Open.Connection** is NULL, and **Open.Durable** is TRUE, the client SHOULD attempt to reconnect to this open, as specified in section [3.2.4.4](#). If the reconnect succeeds, this FSCTL MUST be retried. If it fails, the error code MUST be returned to the application.

If **Open.Connection** is NULL, and **Open.Durable** is FALSE, the client MUST fail this FSCTL operation.

If **Open.Connection** is not NULL and if the client implements SMB 2.1 dialect and **Open.Connection.Dialect** is not equal to "2.100", the client MUST fail the application request with STATUS_NOT_SUPPORTED.

If **Open.Connection** is not NULL, the client MUST format a 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.
- The **MessageId** field is set as specified in section [3.2.4.1.3](#).
- The **SessionId** field is set to **Open.TreeConnect.Session.SessionId**.
- The **TreeId** field is set to **Open.TreeConnect.TreeConnectId**.

The SMB2 IOCTL Request MUST be initialized as follows:

- The **CtlCode** field is set to FSCTL_SRV_READ_HASH.
- 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_READ_HASH request structure that is constructed following the syntax specified in section [2.2.31.2](#) with the client input parameters as follows:
 - The client initializes a SRV_READ_HASH request structure following the syntax specified in section [2.2.31.2](#) using the application provided hash type, hash version, hash retrieval type, length and offset, in bytes.
- The SRV_READ_HASH request structure is copied into the request at InputOffset bytes from the beginning of the SMB2 header.
- The **OutputOffset** field SHOULD [<112>](#) be set to zero.
- The **OutputCount** field is set to 0.
- The **MaxInputResponse** field is set to 0.
- The **MaxOutputResponse** field is set to the maximum number of bytes that the application expects to retrieve.
- The SMB2_0_IOCTL_IS_FSCTL in the Flags field is set to TRUE.

The request MUST be sent to the server, and the response from the server MUST be handled as described in section [3.2.5.14.7](#).

The status of the response MUST be returned to the application.

3.2.4.20.8 Application Requests Resiliency on an Open File

The application provides the following:

- A handle to the **Open** identifying the file to on which to request resiliency.
- The time-out for which the server must hold the handle open on behalf of the client after a network disconnection, in milliseconds.

If the handle is invalid, or if no **Open** referenced by the handle is found, the client MUST return an implementation-specific error code. If the handle is valid and **Open** is found, the client MUST proceed as follows.

If **Open.ResilientHandle** is TRUE, an error MUST be returned to the application.

If **Open.Connection** is NULL, and **Open.Durable** is TRUE, the client SHOULD attempt to reconnect to this open, as specified in section [3.2.4.4](#). If it fails, the error code MUST be returned to the application.

If **Open.Connection** is not NULL and if the client implements SMB 2.1 dialect and **Open.Connection.Dialect** is not equal to "2.100", the client MUST fail the application request with STATUS_NOT_SUPPORTED.

The client MUST set **Open.ResilientTimeout** to the application supplied time-out.

If **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.
- The **MessageId** field is set as specified in section [3.2.4.1.3](#).
- The **SessionId** field is set to **Open.TreeConnect.Session.SessionId**.
- The **TreeId** field is set to **Open.TreeConnect.TreeConnectId**.

The SMB2 IOCTL Request MUST be initialized as follows:

- The **CtlCode** field MUST be set to FSCTL_LMR_REQUEST_RESILIENCY.
- The **FileId** field MUST be set to **Open.FileId**.
- The **InputOffset** field MUST be set to the offset to the **Buffer[]**, in bytes, from the beginning of the SMB2 header.
- The **InputCount** field MUST be set to the size, in bytes, of the [NETWORK_RESILIENCY_REQUEST](#) structure specified in section [2.2.31.3](#)
- A NETWORK_RESILIENCY_REQUEST structure MUST be appended to the request at **InputOffset** bytes from the beginning of the SMB2 header. The **Timeout** field of the NETWORK_RESILIENCY_REQUEST structure MUST be set to the time-out (in milliseconds) provided by the application.
- The **OutputOffset** field SHOULD [<113>](#) be set to zero.
- The **OutputCount** field MUST be set to 0.
- The **MaxInputResponse** field MUST be set to 0.
- The **MaxOutputResponse** field MUST be set to 0.
- SMB2_0_IOCTL_IS_FSCTL in the **Flags** field MUST be set to TRUE.

The request MUST be sent to the server, and the response from the server MUST be handled as described in section [3.2.5.14.9](#).

The status of the response MUST be returned to the application.

3.2.4.20.9 Application Requests Waiting for a Connection to a Pipe

The application provides:

- A handle to the **TreeConnect** identifying the connection to the IPC\$ share.
- The name of the **named pipe**, omitting any prefixes such as "\\pipe\".
- An optional timeout value indicating the maximum amount of time to wait for availability of the pipe, in units of 100 milliseconds.

If the handle is invalid, or if no **TreeConnect** referenced by the tree connect handle is found, the client MUST return an implementation-specific error code locally to the calling application.

If the length of the name of the named pipe is greater than 0xFFFF, the client MUST fail the request and return STATUS_INVALID_PARAMETER to the calling application.

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.
- The **MessageId** field is set as specified in section [3.2.4.1.3](#).
- The **SessionId** field is set to **TreeConnect.Session.SessionId**.
- The **TreeId** field is set to **TreeConnect.TreeConnectId**.

The SMB2_IOCTL Request MUST be initialized as specified in section [2.2.31](#), with the exception of the following values:

- The **CtlCode** field is set to FSCTL_PIPE_WAIT.
- The **FileId** field is set to { 0xFFFFFFFFFFFFFFFF, 0xFFFFFFFFFFFFFFFF }.
- The **MaxInputResponse** field is set to 0.
- The **MaxOutputResponse** field is set to 0.
- SMB2_0_IOCTL_IS_FSCTL is set to TRUE in the **Flags** field.
- The **Buffer** field is set to an FSCTL_PIPE_WAIT Request, as specified in [\[MS-FSCC\]](#) section 2.3.27.
 - **Timeout** is set to the application provided timeout value, or 0 if none was provided.
 - **TimeoutSpecified** is set to TRUE if the application provided a timeout value, or FALSE otherwise.
 - **Name** is set to the name of the named pipe.
 - **NameLength** is set to the length, in bytes, of the **Name** field.
- 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 **Buffer** field.

The request MUST be sent to the server.

3.2.4.21 Application Requests Unlocking of an Array of Byte Ranges

The application provides:

- A handle to the **Open** identifying a file.
- An array of byte ranges to unlock. For each range, the application provides:
 - A starting offset, in bytes.
 - A length, in bytes.

If the handle is invalid, or if no **Open** referenced by the handle is found, the client MUST return an implementation-specific error code. If the handle is valid and **Open** is found, the client MUST proceed as follows.

If **Open.Connection** is NULL, and **Open.Durable** is TRUE, the client SHOULD attempt to reconnect to this open as specified in section [3.2.4.4](#). If the reconnect succeeds, the unlock MUST be retried. If it fails, the error code MUST be returned to the application.

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

If **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.
- The **MessageId** field is set as specified in section [3.2.4.1.3](#).
- The **SessionId** field is set to **Open.TreeConnect.Session.SessionId**.
- The **TreeId** field is set to **Open.TreeConnect.TreeConnectId**.

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 ELEMENT](#) 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.

If **Open.ResilientHandle** is TRUE, the client MUST do the following:

- The client MUST scan through **Open.OperationBuckets** and find an element with its **Free** field set to TRUE. If no such element could be found, an implementation-specific error MUST be returned to the application.
- Let the zero-based array index of the element chosen above be referred to as **BucketIndex**, and let **BucketNumber** = **BucketIndex** + 1.
- Set **Open.OperationBuckets[BucketIndex].Free** = FALSE
- Let the **SequenceNumber** of the element chosen above be referred to as **BucketSequence**.
- The **LockSequence** field of the SMB2 lock request MUST be set to (BucketNumber << 4) + **BucketSequence**.
- Increment the **SequenceNumber** of the element chosen above using MOD 16 arithmetic.

The request MUST be sent to the server, and the response from the server MUST be handled as described in section [3.2.5.13](#).

The status of the response MUST be returned to the application.

3.2.4.22 Application Requests Closing a Share Connection

The application provides a handle to the **TreeConnect**. If the handle is invalid, or if no **TreeConnect** referenced by the handle is found, the client MUST return an implementation-specific error code locally to the calling application. If the handle is valid and a **TreeConnect** is found, the client MUST enumerate all open files on **TreeConnect.Session.Connection.OpenTable** and close those Opens where **Open.TreeConnect** matches the **TreeConnect** by issuing an SMB2 CLOSE as specified in section [3.2.4.4](#).

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.
- The **MessageId** field is set as specified in section [3.2.4.1.3](#).
- The **SessionId** field is set to **TreeConnect.Session.SessionId**.
- The **TreeId** field is set to **TreeConnect.TreeConnectId**.

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.

3.2.4.23 Application Requests Terminating an Authenticated Context

The application provides a handle to the **Session**. If the handle is invalid, or if no **Session** referenced by the handle is found, the client MUST return an implementation-specific error code locally to the calling application. If the handle is valid and a **Session** is found, the client MUST close all tree connects in the **Session.TreeConnectTable**, as specified in section [3.2.4.22](#).

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.
- The **MessageId** field is set as specified in section [3.2.4.1.3](#).
- The **SessionId** field is set to **Session.SessionId**.

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.

3.2.4.24 Application Requests Canceling an Operation

The application provides the **CancelId** of the operation that is to be canceled.

The client MUST enumerate all connections in the **ConnectionTable** and look up a **Request** in **Connection.OutstandingRequests** where **Request.CancelId** matches the application-supplied **CancelId**. If there is a match, the client performs the following:

The client initializes an [SMB2 CANCEL Request](#) following the syntax specified in section [2.2.30](#). The [SMB2 header](#) is initialized as follows:

- The **Command** field MUST be set to SMB2 CANCEL.

- The **MessageId** field SHOULD [<114>](#) be 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 **SessionId** field SHOULD [<115>](#) be set to 0.

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, and no status is returned to the caller. If the server has not processed the original request, it MUST cancel that pending request.

3.2.4.25 Application Requests the Session Key for an Authenticated Context

The application provides a handle to an **Open** established on the session of interest. If the handle is invalid or if no **Open** referenced by the handle is found, the client MUST return an implementation-specific error code locally to the calling application. If the handle is valid and an **Open** is found and the **Open.TreeConnect** is not NULL, the client MUST return the **Open.TreeConnect.Session.SessionKey** to the calling application. If the handle is valid and an **Open** is found and the **Open.TreeConnect** is NULL, the client MUST return an implementation-specific error code locally to the calling application.

3.2.4.26 Application Requests Number of Opens on a Tree Connect

The application provides a handle to the **TreeConnect** representing the share to be queried.

The client MUST query the total number of opens on the **TreeConnect** by looking up the **Connection.OpenTable** where **Open.TreeConnect** matches the application-supplied **Client.TreeConnect**, and return the matching count to the calling application.

3.2.5 Processing Events and Sequencing Rules

The SMB 2 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 MessageId is 0xFFFFFFFFFFFFFFFF, no verification is necessary.

If the [SMB2 header](#) of the response has SMB2_FLAGS_SIGNED set in the **Flags** field, the client MUST verify the signature as follows:

The client 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 client 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 response 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. If the message is an interim response or an SMB2_OPLOCK_BREAK notification, signing validation MUST NOT occur. Otherwise, the client MUST look up the session in the **Connection.SessionTable** using the **SessionId** in the SMB2 header of the response. If the session is found, the **Session.ShouldSign** is equal to TRUE, the message is not an interim response, and the message is not an SMB2_OPLOCK_BREAK notification, 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. [<116>](#)

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 [3.2.4.1.6](#). 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 [3.2.4.23](#).

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 [<117>](#) also disconnect the connection.

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 [<118>](#) fail the application request with an implementation-specific error that indicates an invalid network response was received.

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.

If the **SecurityMode** field in the SMB2 header of the response does not have the SMB2_NEGOTIATE_SIGNING_ENABLED bit set, the client MUST return STATUS_INVALID_NETWORK_RESPONSE.

If the **SecurityMode** field in the SMB2 header of the response has the SMB2_NEGOTIATE_SIGNING_ENABLED bit set, 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**. [<119>](#) The client MUST store the received security buffer described by **SecurityBufferOffset** and **SecurityBufferLength** into **Connection.GSSNegotiateToken**.

If the **SecurityMode** field in the SMB2 header of the response has the SMB2_NEGOTIATE_SIGNING_REQUIRED bit set, the client MUST set **Connection.RequireSigning** to TRUE.

If the **DialectRevision** in the [SMB2 NEGOTIATE Response](#) is 0x02FF, the client MUST issue a new SMB2 NEGOTIATE request as described in section [3.2.4.2.2](#). Otherwise, the client MUST proceed as follows.

If the client supports SMB 2.1, the client MUST perform the following:

- The client MUST store the returned dialect in **Connection.Dialect**.
- If **SMB2_GLOBAL_CAP_LEASING** is set in the **Capabilities** field of the SMB2 NEGOTIATE Response, the client MUST set **Connection.SupportsLeasing** to TRUE. Otherwise, it MUST be set to FALSE.
- If **SMB2_GLOBAL_CAP_LARGE_MTU** is set in the **Capabilities** field of the SMB2 NEGOTIATE Response, the client MUST set **Connection.SupportsMultiCredit** to TRUE. Otherwise, it MUST be set to FALSE.

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.UserCredentials** MUST be set to the OS-specific entity that identifies the credentials that were used to authenticate to the server.
- **Session.SessionKey** MUST be set as described in section [3.2.1.3](#) using the value queried from the GSS 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 generate a handle for the **Session**, and return the handle to the 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 **MessageId** field is set as specified in section [3.2.4.1.3](#).
- 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:

- If RequireMessageSigning is TRUE, the client MUST set the SMB2_NEGOTIATE_SIGNING_REQUIRED bit in the **SecurityMode** field.

If RequireMessageSigning is FALSE, the client MUST set the SMB2_NEGOTIATE_SIGNING_ENABLED 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 **MessageId** field is set as specified in section [3.2.4.1.3](#).
- 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:

- If [RequireMessageSigning](#) is TRUE, the client MUST set the SMB2_NEGOTIATE_SIGNING_REQUIRED bit in the **SecurityMode** field.

If RequireMessageSigning is FALSE, the client MUST set the SMB2_NEGOTIATE_SIGNING_ENABLED 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 **Connection.SessionTable** and freed. The client MUST return success to the application that requested the authenticated context termination, and it MUST invalidate the **Session** handle.

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, and locate the request message in **Connection.OutstandingRequests** using the **MessageId**. The client MUST allocate a tree connect object and insert it into the **Session.TreeConnectTable**. The tree connect is initialized as follows:

- **TreeConnect.TreeConnectId** MUST be set to the **TreeId** that is received in the SMB2 header of the response.
- **TreeConnect.Session** MUST be set to the session that is looked up using **SessionId** from the SMB2 header of the response.
- **TreeConnect.IsDfsShare** MUST be set to TRUE, if the SMB2_SHARE_CAP_DFS bit is set in the **Capabilities** field of the response.
- **TreeConnect.ShareName** MUST be set to the share name, taken from the share path in the request message.

The client MUST generate a **handle** for the **TreeConnect** and return the handle and share type received in the response to the application that initiated the connection to the share. The client sets the share type based on **ShareType** in the response.

Share type	ShareType
"Disk Share"	SMB2_SHARE_TYPE_DISK 0x01
"Named Pipe"	SMB2_SHARE_TYPE_PIPE 0x02
"Printer Share"	SMB2_SHARE_TYPE_PRINT 0x03

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, and it MUST invalidate the **TreeConnect** handle.

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.

The client MUST locate the corresponding request in **Connection.OutstandingRequests** using the **MessageId** field of the SMB2 header. If the request is for a new create operation, then the processing MUST continue as specified below.

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.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.ResilientHandle** MUST be set to FALSE.
- **Open.LastDisconnectTime** MUST be set to zero.
- If **TreeConnect.IsDfsShare** is TRUE, **Open.FileName** MUST be initialized with the name from the **Buffer** field of the request.
- If **TreeConnect.IsDfsShare** is FALSE, **Open.FileName** MUST be initialized with the concatenation of **Connection.ServerName**, **TreeConnect.ShareName**, and the name from the **Buffer** field of the request, joined with pathname separators (example: server\share\path).

Open.OperationBuckets MUST be initialized as follows:

Open.OperationBuckets[x].Free = TRUE for all x = 0 through 63;

Open.OperationBuckets[x].SequenceNumber = 0 for all x = 0 through 63.

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**. If the client implements the SMB 2.1 dialect and **Connection.SupportsLeasing** is TRUE, the client MUST locate the File corresponding to **Open.FileName** in the **GlobalFileTable**. If no **File** is found, the client MUST create a new **File** entry and add it to the **GlobalFileTable** and assign a new **File.LeaseKey**, as specified in section [3.2.1.5](#), to the entry. **File.OpenTable** MUST be initialized to an empty table and **File.LeaseState** MUST be initialized to SMB2_LEASE_NONE. The client MUST insert the Open into **File.OpenTable**.

The client MUST generate a handle for the **Open**, and it MUST return success and the generated handle to the calling application.

3.2.5.7.1 SMB2_CREATE_DURABLE_HANDLE_RESPONSE Create Context

If the [SMB2_CREATE_DURABLE_HANDLE_RESPONSE](#) context is present, the client MUST set **Open.Durable** to TRUE. Otherwise, the client MUST set **Open.Durable** to FALSE.

3.2.5.7.2 SMB2_CREATE_QUERY_MAXIMAL_ACCESS_RESPONSE Create Context

If the [SMB2_CREATE_QUERY_MAXIMAL_ACCESS_RESPONSE](#) context is present, and **QueryStatus** in the SMB2_CREATE_QUERY_MAXIMAL_ACCESS_RESPONSE context is STATUS_SUCCESS, the client MUST return the **MaximalAccess** received in the context to the calling application.

3.2.5.7.3 SMB2_CREATE_QUERY_ON_DISK_ID Create Context

If the SMB2_CREATE_QUERY_ON_DISK_ID context is present, the client MUST return the **DiskIDBuffer** received in the context to the calling application.

3.2.5.7.4 SMB2_CREATE_RESPONSE_LEASE Create Context

If the client implements the SMB 2.1 dialect and an SMB2_CREATE_RESPONSE_LEASE create context is present in the [SMB2_CREATE response](#) returned from the server, it MUST do the following:

- If **Connection.SupportsLeasing** is FALSE, the client MUST fail the create request from the application.
- The client MUST locate the file corresponding to **Open.FileName** in the **GlobalFileTable** and copy the **LeaseState** in the response to **File.LeaseState**.

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 caller of section [3.2.4.4](#) that initiated the open reestablishment operation.

The client MUST locate the corresponding request in **Connection.OutstandingRequests** using the **MessageId** field of the SMB2 header. If the request is for an open reestablishment, then the processing MUST continue as specified below using the **Open** associated with this request in section [3.2.4.4](#).

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 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**. If the client implements the SMB 2.1 dialect and **Connection.SupportsLeasing** is TRUE, the client MUST locate the File corresponding to **Open.FileName** in the **GlobalFileTable**. If no **File** is found, the client MUST create a new **File** entry and add it to the **GlobalFileTable** and assign a new **File.LeaseKey**, as specified in section [3.2.1.5](#), to the entry. **File.OpenTable** MUST be initialized to an empty table and **File.LeaseState** MUST be initialized to SMB2_LEASE_NONE. The client MUST insert the Open into **File.OpenTable**.

If the client implements the SMB 2.1 dialect and an SMB2_CREATE_RESPONSE_LEASE create context is present in the SMB2 CREATE response returned from the server, the client MUST do the following:

- If **Connection.SupportsLeasing** is FALSE, the client MUST return an error to the caller of section [3.2.4.4](#) that initiated the open reestablishment operation.
- Otherwise, the client MUST copy the **LeaseState** in the response to **File.LeaseState**.

The client MUST return success to the caller of section [3.2.4.4](#) that initiated the open reestablishment operation.

3.2.5.9 Receiving an SMB2 CLOSE Response

If the **Status** field of the [SMB2 header](#) of the response indicates an error, then the client MUST return the received status code to the calling application.

The client MUST locate the Open in the **Connection.OpenTable** using the **FileId** in the SMB2 header of the response.

If the client implements the SMB 2.1 dialect and **Connection.SupportsLeasing** is TRUE, the client MUST locate the **File** in the **GlobalFileTable** by looking up **Open.FileName**. The client MUST delete the **Open** from the **File.OpenTable**. If all opens in **File.OpenTable** are deleted, then the entry for this **File** MUST be deleted from the **GlobalFileTable**, and the **File** object MUST be freed.

The open object MUST be removed from the **Connection.OpenTable** and freed.

The client MUST return success to the application that requested that the file or named pipe be closed, and it MUST invalidate the **Open** handle.

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 look up the corresponding request by looking up **Connection.OutstandingRequests** using the **MessageId** from the SMB2 header. If the **LockSequence** field in the request is nonzero, then the client MUST do the following:

1. From the **LockSequence** field in the SMB2 lock or SMB2 unlock request (constructed in sections [3.2.4.19](#) and [3.2.4.21](#)) that was looked up as described in the previous paragraph, compute

```
BucketNumber = (LockSequence >> 4)
BucketIndex = BucketNumber - 1
```


2. Set **Open.OperationBuckets[BucketIndex].Free** = TRUE;

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

If the **OutputCount** field in an [SMB2 IOCTL Response](#) is 0, the **OutputOffset** field SHOULD [<120>](#) be ignored by the client.

[IOCTL](#)-specific processing is specified in the following sections.

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 **OutputCount** into the buffer that is provided by the calling application for receiving the response output buffer. The client MUST return success and **OutputCount** 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 **OutputCount** into the buffer that is provided by the calling application for receiving the response output buffer. The client MUST return success and **OutputCount** to the application.

3.2.5.14.3 Handling a Server-Side Data Copy Response

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 MUST return a result of **STATUS_INVALID_PARAMETER** to the application and SHOULD also return the values in the accompanying [SRV_COPYCHUNK_RESPONSE](#) that indicate the maximum limits the server supports for server side copy operations.

Otherwise, 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, ignoring any accompanying [SRV_COPYCHUNK_RESPONSE](#).

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 **OutputCount** into the buffer that is provided by the calling application for receiving the response output buffer. The client MUST return success and the **OutputCount** 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 the **OutputOffset** and **OutputCount** into the buffer that is provided by the calling application for receiving the response output buffer. The client MUST return success and the **OutputCount** 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 **OutputCount** 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 **OutputCount** into the buffer that is provided by the calling application for receiving the response output buffer. The client MUST return success and the **OutputCount** to the application.

3.2.5.14.7 Handling a Content Information Retrieval 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 **OutputCount** to the application.

3.2.5.14.8 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 **OutputCount** 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 **OutputCount** to the application.

3.2.5.14.9 Handling a Resiliency 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 perform the following steps:

1. The client SHOULD [<121>](#) setup periodic probing of the connection to detect an unresponsive or dead server or a broken TCP connection.
2. The client MUST set **Open.ResilientHandle** and **Open.Durable** to TRUE.
3. The status of the operation MUST be returned to the application.

3.2.5.14.10 Handling a Pipe Wait Response

The client MUST return the **Status** field of the [SMB2 header](#) of the response to the calling 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 either **STATUS_BUFFER_TOO_SMALL** or **STATUS_INFO_LENGTH_MISMATCH** 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 indicates 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, underlying object store, or named pipe. This applies for requests to set file information, underlying object store 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.

If the client implements the SMB 2.1 dialect and **Connection.SupportsLeasing** is TRUE, the client MUST use the **StructureSize** member in the [SMB2 OPLOCK_BREAK notification](#) to differentiate between an oplock break notification and a lease break notification as described in [2.2.25](#).

3.2.5.19.1 Receiving an Oplock Break Notification

The client MUST locate the open in the **Connection.OpenTable** using the **FileId** in the [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 Oplock Break Notification.

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, specified as follows.

The client MAY [<122>](#) 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.

If the client is required to send an oplock break acknowledgment, it MUST construct a request following the syntax that is specified in section [2.2.24.1](#). The SMB2 header is initialized as follows:

- **Command** MUST be set to SMB2_OPLOCK_BREAK.

- The **MessageId** field is set as specified in section [3.2.4.1.3](#).
- The client MUST set SessionId to **Open.TreeConnect.Session.SessionId**.
- The client MUST set **TreeId** to **Open.TreeConnect.TreeConnectId**.

The Oplock Break Acknowledgment 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 a Lease Break Notification

If the client implements the SMB 2.1 dialect and **Connection.SupportsLeasing** is TRUE, the client MUST perform the following:

- The client MUST locate the file in the **GlobalFileTable** using the **LeaseKey** in the [Lease Break Notification](#). If a file is not found, no further processing is required.
- If a File entry is located, the client MUST take action based on the **File.LeaseState** and the new LeaseState that is received in the Lease Break Notification.
- If **File.LeaseState** includes SMB2_LEASE_WRITE_CACHING and the new LeaseState does not include SMB2_LEASE_WRITE_CACHING, the client MUST flush any cached data associated with this file by issuing one or more SMB2 WRITE requests as described in [3.2.4.7](#). It MUST also flush out any cached byte-range locks it has on the file by enumerating the **File.OpenTable** and for each open, send the cached byte-range locks by issuing SMB2 LOCK requests as described in [3.2.4.19](#).
- If **File.LeaseState** includes SMB2_LEASE_READ_CACHING and the new LeaseState does not include SMB2_LEASE_READ_CACHING, the client MUST purge any cached data.
- If **File.LeaseState** includes SMB2_LEASE_HANDLE_CACHING and the new LeaseState does not include SMB2_LEASE_HANDLE_CACHING, the client MUST enumerate all handles in **File.OpenTable** and close any cached handles that have already been closed by the application. The close process is described in [3.2.4.5](#).
- The new LeaseState granted by the server in the Lease Break Notification MUST be copied to **File.LeaseState**.
- If a lease acknowledgment is required by the server as indicated by the SMB2_NOTIFY_BREAK_LEASE_FLAG_ACK_REQUIRED bit in the **Flags** field of the Lease Break Notification, the client MUST send a [Lease Break Acknowledgment](#) request described as follows.
 - If all open handles on this file are closed (that is, **File.OpenTable** is empty for this file), the client SHOULD consider it as an implicit acknowledgment of the lease break. No explicit acknowledgment is required.
 - The client MUST construct a Lease Break Acknowledgment request following the syntax specified in [2.2.24.2](#). The LeaseKey in the request MUST be set to **File.LeaseKey** and the LeaseState in the request MUST be set to **File.LeaseState**.

- The client MUST choose an Open from among the remaining opens in **File.OpenTable** and it MUST be used to send the acknowledgment to the server, via the connection identified by **Open.Connection**.
- The [SMB2 header](#) is initialized as follows:
 - **Command** MUST be set to SMB2 OPLOCK_BREAK.
 - The **MessageId** field is set as specified in section [3.2.4.1.3](#).
 - The client MUST set **SessionId** to **Open.TreeConnect.Session.SessionId**.
 - The client MUST set **TreeId** to **Open.TreeConnect.TreeConnectId**.

3.2.5.19.3 Receiving an Oplock Break Acknowledgment Response

No processing is required for this response.

3.2.5.19.4 Receiving a Lease 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**.

The client MAY [<123>](#) choose any time-out it requires based on local policy, the type of request, and network characteristics.

Clients MAY [<124>](#) choose to extend the time-out if the server returns STATUS_PENDING, which indicates that the operation is being processed asynchronously and may take a long time to complete, as specified in section [3.2.5.1.4](#).

If **Request.Timestamp** plus the time-out interval exceeds the current time, the client MUST process the request as if it received a failure response from the server and the client SHOULD [<125>](#) return an implementation-specific error to the calling application.

The client MAY [<126>](#) choose to disconnect the connection as well.

3.2.6.2 Idle Connection Timer Event

When the Idle Connection timer expires, the client MUST scan through the global **ConnectionTable** (defined in section [3.2.1.1](#)). For each connection, if **Connection.OpenTable** is empty and the idle time-out has expired, the client MUST tear down the Connection and all associated Sessions and Tree Connects, in the manner specified in section [3.2.7.1](#). The client is not required to explicitly send [SMB2 LOGOFF](#) and [SMB2 TREE_DISCONNECT](#) requests to the server because the teardown of the connection will implicitly result in the teardown of all server Sessions and Tree Connects on the connection, as specified in section [3.3.7.1](#).

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**, the client MUST perform the following actions:

- If the client implements the SMB 2.1 dialect and **Connection.SupportsLeasing** is TRUE, the client MUST locate the **File** in the **GlobalFileTable** by looking up **Open.FileName**. The client MUST delete the **Open** from the **File.OpenTable**. If all opens in **File.OpenTable** are deleted, then the entry for this **File** MUST be deleted from the **GlobalFileTable**, and the **File** object MUST be freed.
- If **Open.Durable** is not TRUE, the **Open** MUST be removed from the **Connection.OpenTable** and freed, and the handle generated for the **Open** MUST be invalidated.
- 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.
- If **Open.ResilientHandle** is TRUE, the client MUST perform the following steps:
 - Capture the current system time at which the disconnect occurred into **Open.LastDisconnectTime**.
 - Attempt to reestablish the durable open as specified in section [3.2.4.4](#).
 - If the reestablishment fails with a network error, the client MUST retry the reestablishment of the open for at least **Open.ResilientTimeout** milliseconds after **Open.LastDisconnectTime**, before giving up.
 - If the reestablishment of the durable handle fails, **Open.Durable** MUST be set to FALSE, **Open.ResilientHandle** MUST be set to FALSE, the **Open** MUST be removed from **Connection.OpenTable** and the **Open** MUST be freed, and the handle generated for the **Open** MUST be invalidated.

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 and the handle generated for the tree connect MUST be invalidated. Then the session MUST be freed and the handle generated for the session MUST be invalidated.

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. This data model requires elements to be synchronized with the Server Service Remote Protocol [MS-SRVS]. An implementation that uses this data model should observe atomicity requirements in order that the protocols always maintain an identical view of the common data.

This document assumes the SMB 2 Protocol server is a combination of a server and one or more underlying object store(s). However, an implementation may subdivide the server into whatever functional blocks it chooses, including combining them into a single block.

3.3.1.1 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 [3.3.1.2](#), 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** [<127>](#) choose to not grant more credits and keep the valid command window set at { 0 } until the sequence number 0 is received.
- The client's request consumes at least one sequence number for any request except the [SMB2 CANCEL Request](#). If the server implements SMB 2.1 and the request is a multi-credit request, it consumes sequence numbers based on the **CreditCharge** field in the SMB2 header, as specified in [3.3.5.2.2](#).

For the client side of this algorithm, see section [3.2.4.1.6](#).

3.3.1.2 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 held by the client **MUST** be considered as 1 when the connection is established.
- The server **MUST** ensure that the number of credits held by 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** [<128>](#) grant any number of credits up to that which the client requests, or more if required by the preceding rule.

- The server SHOULD<129> grant the client a non-zero value of credits in response to any non-zero value requested, within administratively configured limits. The server MUST grant the client at least 1 credit when responding to SMB2 NEGOTIATE.
- The server MAY<130> vary the number of credits granted to different clients based on quality of service features, such as identity, behavior, or administrator configuration.

3.3.1.3 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 [MS-CIFS] sections 3.2.4.39 and 3.3.5.59.4. 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 [SMB2 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 the maximum number of events that match the **CompletionFilter** of the request 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.

3.3.1.4 Algorithm for Leasing in an Object Store

If the server implements SMB 2.1 and supports leasing, the underlying object store MUST implement an algorithm that permits multiple opens to the same object to share lease state (for valid lease states, see section 3.3.1.12). The algorithm MUST meet the following conditions:

- The algorithm MUST permit a create request from the server to the underlying object store to be accompanied by a global identifier that indicates the unique context for this lease, which will be referred to as the ClientId. The ClientId consists of a ClientGuid combined with a **LeaseKey**.
- The algorithm MUST allow multiple opens to an object that specifies the same ClientId. These opens MUST NOT alter the lease state on an object.
- The algorithm MUST permit three different caching capabilities within a lease: READ, WRITE, and HANDLE, with the following semantics:
 - READ caching permits the SMB client to cache data read from the object. If one of the following operations occurs, the object store MUST request that the server revoke READ caching. The object store is not required to wait for acknowledgment:
 - The file is opened in a manner that overwrites the existing file.
 - Data is written to the file.

- The file size is changed.
- A byte range lock is requested for the file.
- WRITE caching permits the SMB client to cache writes and byte-range locks on an object. If one of the following operations is about to occur, the underlying object store MUST request that the server revoke WRITE caching, and the object store MUST wait for acknowledgment from the server before proceeding with the operation:
 - The file is opened by a local application or via another protocol, or opened via SMB 2 without providing the same ClientId, and requested access includes any flags other than FILE_READ_ATTRIBUTES, FILE_WRITE_ATTRIBUTES, and SYNCHRONIZE.
- HANDLE caching permits one or more SMB clients to delay closing handles it holds open, or to defer sending opens. If one of the following operations is about to occur, the underlying object store MUST request that the server revoke HANDLE caching, and the object store MUST wait for acknowledgment before proceeding with the operation:
 - A file is being opened by a local application, via another protocol, or opened via SMB 2 without providing the same ClientId, and this open will fail with SHARING_VIOLATION due to the existing opens that hold the lease.
 - The file is being renamed or deleted by an open with a different ClientId than the lease was acquired with.
 - The parent directory is being renamed.
- The underlying object store SHOULD request that the server revoke multiple lease state flags at the same time if an operation results in the loss of several caching flags.
- The algorithm SHOULD support the following combinations of caching flags: No caching, Read Caching, Read + Write Caching, Read + Handle Caching, Read + Write + Handle Caching.
- The algorithm MAY [<131>](#) support other combinations of caching flags.
- The algorithm MUST allow a client to flow one or more creates with the same ClientId to the underlying object store during a lease break without blocking the create until the acknowledgment of the lease break is received.
- The algorithm SHOULD allow additional lease state flags on subsequent opens from the same ClientId to permit upgrading the lease state. The algorithm MUST NOT allow the client to release lease state flags on subsequent opens from the same ClientId to downgrade the lease state.
- If the requested lease state is not a superset of the existing lease state flags for this ClientId, then the requested lease state SHOULD be interpreted as the union of the existing lease state and the requested lease state.
- When the underlying object store requests that the server issue a lease break, it MUST also provide a new lease state for the server to pass to the client as part of the lease break packet, based on the local operations that caused the lease break to occur.

3.3.1.5 Global Structures

The server implements the following:

- **ServerStatistics:** Server statistical information. This contains all the members of **STAT_SRV_0** structure as specified in [\[MS-SRVS\]](#) section 2.2.4.39.

- **ServerEnabled:** A Boolean that indicates whether the SMB2 server is accepting incoming connections or requests.
- **ShareList:** A list of available shares for the system. The structure of a share is as specified in section [3.3.1.6](#) and is uniquely indexed by the tuple <Share.ServerName, Share.Name>.
- **GlobalOpenTable:** A table containing all the files opened by remote clients on the server, indexed by **Open.DurableFileId**. The structure of an open is as specified in section [3.3.1.10](#). The table MUST support enumeration of all entries in the table.
- **GlobalSessionTable:** A list of all the active sessions established to this server, indexed by the **Session.SessionId**.
- **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.
- **IsDfsCapable:** A Boolean that, if set, indicates that the server supports the Distributed File System.
- **ServerSideCopyMaxNumberOfChunks:** The maximum number of chunks the server will accept in a server side copy operation.
- **ServerSideCopyMaxChunkSize:** The maximum number of bytes the server will accept in a single chunk for a server side copy operation.
- **ServerSideCopyMaxDataSize:** The maximum total number of bytes the server will accept for a server side copy operation.

If the server implements SMB 2.1, it MUST implement the following:

- **ServerHashLevel:** A state that indicates the caching level configured on the server. It takes any of the following three values:
 - **HashEnableAll:** Indicates that caching is enabled for all shares on the server.
 - **HashDisableAll:** Indicates that caching is disabled for all shares on the server.
 - **HashEnableShare:** Indicates that caching is enabled or disabled on a per-share basis.

If the server implements SMB 2.1 and supports leasing, it MUST implement the following:

- **GlobalLeaseTableList:** A list of all the lease tables as described in [3.3.1.11](#), indexed by the ClientGuid.
- **MaxResiliencyTimeout:** The maximum resiliency time-out in milliseconds, for the **TimeOut** field of NETWORK_RESILIENCY_REQUEST Request as specified in section [2.2.31.3](#).
- **ResilientOpenScavengerExpiryTime:** The time at which the Resilient Open Scavenger Timer, as specified in section [3.3.2.4](#), is currently set to expire.

3.3.1.6 Per Share Structures

The server implements the following:

- **Share.Name:** A name for the shared resource on this server.
- **Share.ServerName:** The **NetBIOS**, fully qualified domain name (FQDN), or textual IPv4 or IPv6 address that the share is associated with. For more information, see [\[MS-SRVS\]](#) section 3.1.1.7.
- **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 [<132>](#) be a device that accepts a file and then processes it in some format, such as a printer.
- **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. [<133>](#)
- **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 the right to access.
- **Share.AllowNamespaceCaching:** A Boolean that, if set, indicates that clients are allowed to cache directory enumeration results for better performance. [<134>](#)
- **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.
- **Share.Type:** The value indicates the type of share. It MUST be one of the values that are listed in [\[MS-SRVS\]](#) section 2.2.2.4.
- **Share.Remark:** A pointer to a null-terminated Unicode UTF-16 string that specifies an optional comment about the shared resource.
- **Share.MaxUses:** The value indicates the maximum number of concurrent connections that the shared resource can accommodate.
- **Share.CurrentUses:** The value indicates the number of current trees connected to the shared resource.
- **Share.ForceLevel2Oplock:** A Boolean that, if set, indicates that the server does not issue exclusive caching rights on this share.
- **Share.HashEnabled:** A Boolean that, if set, indicates that the share supports hash generation for branch cache retrieval of data.

3.3.1.7 Per Transport Connection Structures

The server implements the following:

- **Connection.CommandSequenceWindow:** A list of the sequence numbers that is valid to receive from the client at this time. For more information, see section [3.3.1.1](#).
- **Connection.RequestList:** A list of requests, as specified in section [3.3.1.13](#), that are currently being processed by the server. This list is indexed by the **MessageId** field.
- **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.NegotiateDialect:** A numeric value representing the current state of dialect negotiation between the client and server on this transport connection.
- **Connection.AsyncCommandList:** A list of client requests being handled asynchronously. Each request MUST have been assigned an **AsyncId**.
- **Connection.Dialect:** The dialect of SMB2 negotiated with the client. This value MUST be either "2.002", "2.100", or "Unknown".
- **Connection.ShouldSign:** A Boolean that, if set, indicates that all sessions on this connection (with the exception of anonymous and guest sessions) MUST have signing enabled.
- **Connection.ClientName:** A null-terminated Unicode UTF-16IP address string, or NetBIOS host name of the client machine.
- **Connection.MaxTransactSize:** The maximum buffer size, in bytes, that the 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.
- **Connection.SupportsMultiCredit:** A Boolean indicating whether the connection supports multi-credit operations.
- **Connection.TransportName:** An implementation-specific name of the transport used by this connection.

If the server implements SMB 2.1, it MUST implement the following:

- **Connection.ClientGuid:** An identifier for the client machine.

3.3.1.8 Per Session Structures

The server implements the following:

- **Session.SessionId:** A numeric value that is used as an index in **GlobalSessionTable**, and (transformed into a 64-bit number) is sent to clients as the **SessionId** in the **SMB2 header**.
- **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.IsAnonymous:** A Boolean that, if set, indicates that the session is for an anonymous user.
- **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, 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 and indexed by **Open.FileId**. The server 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, indexed by **TreeConnect.TreeId**. The server 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.
- **Session.Connection:** The connection on which this session was established (see also section [3.3.5.5.1](#)).
- **Session.SessionGlobalId:** A numeric 32-bit value obtained via registration with [\[MS-SRVS\]](#), as specified in [\[MS-SRVS\]](#) section 3.1.6.2.
- **Session.CreationTime:** The time the session was established.
- **Session.IdleTime:** The time the session processed its most recent request.
- **Session.UserName:** The name of the user who established the session.

3.3.1.9 Per Tree Connect Structures

The server implements the following:

- **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 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.
- **TreeConnect.OpenCount:** A numeric value that indicates the number of files that are currently opened on TreeConnect.
- **TreeConnect.TreeGlobalId:** A numeric value obtained via registration with [\[MS-SRVS\]](#), as specified in [\[MS-SRVS\]](#) section 3.1.6.6.
- **TreeConnect.CreationTime:** The time tree connect was established.

3.3.1.10 Per Open Structures

The server implements the following:

- **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.FileGlobalId:** A numeric value obtained via registration with [\[MS-SRVS\]](#), as specified in [\[MS-SRVS\]](#) section 3.1.6.4.
- **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 open is not attached to a session at this time, this value MUST be NULL.
- **Open.TreeConnect:** A reference to the **TreeConnect**, as specified in section [3.3.1.9](#), over which the open was performed. If the open is not attached to a **TreeConnect** at this time, this value MUST be NULL.
- **Open.Connection:** A reference to the connection, as specified in section [3.3.1.7](#), that created this open. If the open is not attached to a connection at this time, this value MUST be NULL.
- **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. For named pipes, **Open.LocalOpen** is shared between the SMB server and RPC server applications which serve RPC requests on a given named pipe. The higher level interfaces described in sections [3.3.4.5](#) and [3.3.4.11](#) require this shared ADM element.
- **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 one of the **OplockLevel** values defined in section [2.2.14](#): **SMB2_OPLOCK_LEVEL_NONE**, **SMB2_OPLOCK_LEVEL_II**, **SMB2_OPLOCK_LEVEL_EXCLUSIVE**, **SMB2_OPLOCK_LEVEL_BATCH**, or **OPLOCK_LEVEL_LEASE**.
- **Open.OplockState:** The current oplock state of the file. This value MUST be Held, Breaking, or None.
- **Open.OplockTimeout:** The time 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 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 open. This allows the server to determine if a caller that is trying to reestablish a durable open is allowed to do so. If the server implements SMB 2.1 and supports resiliency, this value is also used to enforce security during resilient open reestablishment.
- **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.
- **Open.CurrentEaIndex:** For extended attribute information, this value indicates the current location in an extended attribute information list and allows for the continuing of an enumeration across multiple requests.
- **Open.CurrentQuotaIndex:** For quota queries, this value indicates the current index in the quota information list and allows for the continuation of an enumeration across multiple requests.
- **Open.LockCount:** A numeric value that indicates the number of locks that are held by current open.
- **Open.PathName:** A variable-length string that contains the Unicode path name that the open is performed on.

If the server implements SMB 2.1 and supports leasing, it **MUST** implement the following:

- **Open.Lease:** The lease associated with this open, as defined in [3.3.1.12](#). This value **MUST** point to a valid lease, or be set to NULL.
- **Open.IsResilient:** A Boolean that indicates whether this open has requested resilient operation.
- **Open.ResiliencyTimeout:** A time-out value that indicates how long the server will hold the file open after a disconnect before releasing the open.
- **Open.ResilientOpenTimeout:** A time value that indicates when a handle that has been preserved for resiliency will be closed by the system if a client has not reclaimed it.
- **Open.LockSequenceArray[]:** An array of lock sequence entries (each of size 1 byte) that have been successfully processed by the server for resilient opens. The size of this array is implementation-dependent. [<135>](#)

3.3.1.11 Per Lease Table Structures

If the server implements SMB 2.1 and supports leasing, it implements the following:

- **LeaseTable.ClientGuid:** A global identifier to associate which connections **MUST** use this LeaseTable.
- **LeaseTable.LeaseList:** A list of lease structures, as defined in section [3.3.1.12](#), indexed by **LeaseKey**.

3.3.1.12 Per Lease Structures

If the server implements SMB 2.1 and supports leasing, it implements the following:

- **Lease.LeaseKey:** A global identifier for this lease.
- **Lease.FileName:** The name of the file backing this lease.
- **Lease.LeaseState:** The current state of the lease as indicated by the underlying object store. This value **MUST** be a combination of the flags described in section [2.2.13.2.8](#) for "LeaseState". For the remainder of section [3.3](#), these will be referred to as follows:

Lease State	Abbreviated Name
0	NONE
SMB2_LEASE_READ_CACHING	R
SMB2_LEASE_READ_CACHING SMB2_LEASE_WRITE_CACHING	RW
SMB2_LEASE_READ_CACHING SMB2_LEASE_HANDLE_CACHING	RH
SMB2_LEASE_READ_CACHING SMB2_LEASE_WRITE_CACHING SMB2_LEASE_HANDLE_CACHING	RWH

- **Lease.BreakToLeaseState:** The state to which the lease is breaking. This value MUST be a combination of the flags described in section [2.2.13.2.8](#) for "**LeaseState**". For the remainder of section [3.3](#), these will be referred to as described in the table above.
- **Lease.LeaseBreakTimeout:** The time value that indicates when a lease that is breaking and has not received a [Lease Break Acknowledgment](#) from the client will be acknowledged by the server to the underlying object store.
- **Lease.LeaseOpens:** The list of opens associated with this lease.
- **Lease.Breaking:** A Boolean that indicates if a lease break is in progress.

3.3.1.13 Per Request Structures

The server implements the following:

- **Request.MessageId:** The value of the **MessageId** field from the [SMB2 Header](#) of the client request.
- **Request.AsyncId:** An asynchronous identifier generated for an Asynchronous Operation, as specified in section [3.3.4.2](#). The identifier MUST uniquely identify this **Request** among all requests currently being processed asynchronously on a specified SMB2 transport connection. If the request is not being processed asynchronously, this value MUST be set to zero.
- **Request.CancelRequestId:** An implementation-dependent identifier generated by the server to support cancellation of pending requests that are sent to the object store. The identifier MUST be unique among all requests currently being processed by the server and all object store operations being performed by other server applications. [<136>](#)
- **Request.Open:** A reference to an **Open** of a file or named pipe, as specified in section [3.3.1.10](#). If the request is not associated with an **Open** at this time, this value MUST be NULL.

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. <137>](#) If the server implements SMB 2.1, this timer

MUST also be used to control the time a server waits for a Lease Break Acknowledgment from the client (as specified in section [2.2.24.2](#)).

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. [<138>](#) The server MUST keep the durable handle active for at least this amount of time, except in the cases of an oplock break indicated by the object store as specified in section [3.3.4.6](#), administrative actions, or resource constraints.

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 [<139>](#) schedule this timer such that sessions are expired in a timely manner.

3.3.2.4 Resilient Open Scavenger Timer

This timer controls the amount of time the server keeps a resilient handle active after the underlying transport connection to the client is lost. This value is not a constant but set based on the time-out requested by the client as specified in section [3.3.5.15.9](#). The server MUST keep the resilient handle active for that amount of time except in cases of administrative actions or resource constraints.

3.3.3 Initialization

The server MUST implement the following:

- All the members in **ServerStatistics** MUST be set to zero.
- **ServerEnabled** MUST be set to FALSE.
- **GlobalOpenTable** MUST be set to an empty table.
- **GlobalSessionTable** MUST be set to an empty table.
- **ServerGuid** MUST be set to a newly generated GUID.
- **ConnectionList** MUST be set to an empty list.
- **ServerStartTime** MUST be set to the time at which the SMB2 server was started.
- **IsDfsCapable** MUST be set to FALSE.
- **ServerSideCopyMaxNumberOfChunks** MUST be set to an implementation-specific [<140>](#) default value.
- **ServerSideCopyMaxChunkSize** MUST be set to an implementation-specific [<141>](#) default value.
- **ServerSideCopyMaxDataSize** MUST be set to an implementation-specific [<142>](#) default value.
- **ShareList** MUST be set to an empty list.

If the server implements SMB 2.1, it MUST initialize the following:

- **ServerHashLevel** MUST be set to an implementation-specific [<143>](#) default value.

If the server implements SMB 2.1 and supports leasing, the server MUST implement the following:

- **GlobalLeaseTableList** MUST be set to an empty list.
- **MaxResiliencyTimeout** SHOULD [<144>](#) be set to an implementation-specific default value.

The server MUST notify the completion of its initialization to the server service by invoking the event as specified in [\[MS-SRVS\]](#) section 3.1.6.14, providing the string "SMB2" as input parameter.

3.3.4 Higher-Layer Triggered Events

The SMB 2 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 or lease 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. For every outgoing message, the server MUST calculate the total number of bytes in the message and update the values of **ServerStatistics.sts0_bytessent_low** and **ServerStatistics.sts0_bytessent_high**.

3.3.4.1.1 Signing the Message

If the message is a response to a message that contained a nonzero **SessionId** in the **SMB2 header** field, and either the session identified by that **SessionId** in **GlobalSessionTable** has **Session.ShouldSign** equal to TRUE or the request was signed by the client, and the response is neither an error response nor an interim response to an asynchronously processed request, then it MUST be signed as specified below.

If the response is an [SMB2 OPLOCK BREAK](#) or an interim response to an asynchronously processed request, then the response SHOULD [<145>](#) be signed.

Otherwise, the message MUST NOT be signed.

If signing is required, 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 [3.3.1.1](#), 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 [3.3.1.2](#). The server MUST then place the number of credits granted in the **CreditResponse** field in the SMB2 header of the response.

The server consumes one credit for any request except for the [SMB2_CANCEL Request](#). If the server implements SMB 2.1 and the request is a multi-credit request, the server MUST consume multiple credits as specified in section [3.3.5.2.2](#). To maintain the same number of credits already granted, the server returns a value equal to the number of credits consumed by this command. To reduce or increase the number of credits granted, the server respectively returns a value less than or greater than the number of credits consumed by this command.

For an asynchronously processed request, any credits to be granted MUST be granted in the interim response, as specified in section [3.3.4.2.<146>](#)

3.3.4.1.3 Sending Compounded Responses

The server MAY [<147>](#) compound responses to the client.

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. The server MAY [<148>](#) grant credits separately on each request in the compounded chain. Then the entire response chain MUST be sent to the client as a single submission to the underlying transport. 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 [<149>](#) choose to send an interim response for any request that is received. It SHOULD [<150>](#) 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 [<151>](#) choose to fail that operation with STATUS_INSUFFICIENT_RESOURCES.

An interim response indicates to the client that the request has been received and a full response will come later. As noted in section [3.3.4.1.1](#), it is not mandatory for the server to sign an interim response.

To send an interim response for a request, the server MUST generate an asynchronous identifier for it, and **Request.AsyncId** MUST be set to this asynchronous identifier.

- 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.
- The identifier MUST be nonzero.

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 [<152>](#) fail requests in a compound chain that try to go async.
- 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 set the **CreditResponse** field to the number of credits the server chooses to grant for this request, as specified in section [3.3.1.2](#).

It MUST append an [SMB2 ERROR Response](#) following the SMB2 header, as specified in section [2.2.2](#), with a **ByteCount** of zero. 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 interim response. It MUST also set the SMB2_FLAGS_ASYNC_COMMAND bit in the **Flags** field of the SMB2 header. The server MUST set **CreditResponse** in the SMB2 header to 0, as credits were granted on the interim response. 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 a Success Response

When the server responds with a success to any command sent by the client, the response message MUST be constructed as specified in this section.

The server MUST fill in the [SMB2 header](#) of the success 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 status code provided.
- The **NextCommand** field MUST be set to zero. 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.
- The SMB2_FLAGS_SERVER_TO_REDIR bit MUST be set in the **Flags** field of the SMB2 header.
- If the request was 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.
- If the request was not handled asynchronously, the server MUST set the **CreditResponse** field to the number of credits the server chooses to grant the request, as specified in section [3.3.1.2](#). If the request was handled asynchronously, the server MUST set the **CreditResponse** field to 0.

Any other additional changes to the header will be made on a command-specific basis.

The information that follows the SMB2 header is command-specific, as specified in section [3.3.5](#). This response MUST be sent to the client and the request MUST be removed from **Connection.RequestList** and freed.

3.3.4.4 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_MORE_PROCESSING_REQUIRED in an [SMB2 SESSION SETUP Response](#) specified in section [2.2.6](#).
- STATUS_BUFFER_OVERFLOW in an [SMB2 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.<153>](#)
- STATUS_BUFFER_OVERFLOW in an [SMB2 READ Response](#) on a named pipe specified in section [2.2.20](#).
- Any status other than STATUS_SUCCESS in a FSCTL_SRV_COPYCHUNK or FSCTL_SRV_COPYCHUNK_WRITE Response, when returning an [SRV_COPYCHUNK RESPONSE](#) as described in section [2.2.32.1](#).
- STATUS_NOTIFY_ENUM_DIR in an [SMB2 CHANGE NOTIFY Response](#) specified in section [2.2.36](#).

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 was 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.
- If the request was not handled asynchronously, the server **MUST** set the **CreditResponse** field to the number of credits the server chooses to grant the request, as specified in section [3.3.1.2](#). If the request was handled asynchronously, the server **MUST** set the **CreditResponse** field to 0.

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 and the request **MUST** be removed from **Connection.RequestList** and freed.

3.3.4.5 Server Application Requests Session Key of the Client

An application running on the server issues a query for a session key, specifying the **LocalOpen** to a named pipe that has been opened by the SMB2 server on behalf of the remote client. The application also provides a 16-byte buffer to receive the session key.

The server MUST cycle through the entries in the **GlobalOpenTable** and locate the Open for which **Open.LocalOpen** matches the provided **LocalOpen**. If no Open is found, the request MUST be failed with STATUS_OBJECT_NAME_NOT_FOUND.

If **Open.Connection** is NULL, the request MUST be failed with STATUS_NO_TOKEN.

If **Open.TreeConnect.Share.Name** is not equal to "IPC\$" (indicating that the open is not a named pipe), the request SHOULD be failed with STATUS_ACCESS_DENIED.

Otherwise, the server MUST return the 16-byte session key specified by **Open.TreeConnect.Session.SessionKey** to the caller.

3.3.4.6 Object Store Indicates an Oplock Break

The underlying object store on the local resource indicates the breaking of an opportunistic lock, specifying the **LocalOpen** and the new oplock level, a status code of the oplock break, and optionally expects the new oplock level in return. The new oplock level MUST be either SMB2_OPLOCK_LEVEL_NONE or SMB2_OPLOCK_LEVEL-II. The conditions under which each oplock level is to be indicated are described in [\[MS-FSA\]](#) section 3.1.5.17.3.

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 NULL and the oplock level is not equal to SMB2_OPLOCK_LEVEL_BATCH, the server MUST close the Open as specified in section [3.3.4.17](#). The server MUST then complete the oplock break call with SMB2_OPLOCK_LEVEL_NONE as the new oplock level.

If **Open.Connection** is not NULL, the server MUST construct an [Oplock Break Notification](#) following the syntax specified in section [2.2.23.1](#) to send back to the client. The server MUST set the **Command** in the [SMB2 header](#) to SMB2_OPLOCK_BREAK, the **MessageId** to 0xFFFFFFFFFFFFFFFF, and MUST set the **ProcessId** to 0. The server MUST set the SessionId and TreeId in the SMB2 header to 0. 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 and set **Open.OplockTimeout** to the current time plus an implementation-specific default value in milliseconds. [<154>](#) 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](#).

3.3.4.7 Object Store Indicates a Lease Break

The underlying object store indicates the breaking of a lease by specifying the **ClientGuid**, the **LeaseKey**, and the new lease state. The new lease state MUST be one of NONE, R, RW, and RH.

When the underlying object store indicates the lease break, the server MUST locate the Lease Table by performing a lookup in **GlobalLeaseTableList** using the provided **ClientGuid** as the lookup key, and then locate the Lease entry by performing a lookup in the **LeaseTable.LeaseList** using the provided **LeaseKey** as the lookup key.

If no entry is found, the server MUST NOT generate a [Lease Break Notification](#). Instead, the server MUST complete the lease break call from the underlying object store with "NONE" as the new lease state, and take no further action.

If a lease entry is found, the server MUST check the state of **Open.Connection** for all Opens in **Lease.LeaseOpens**. If **Open.Connection** is NULL and **Lease.BreakToLeaseState** does not contain SMB2_LEASE_HANDLE_CACHING and **Open.IsDurable** is TRUE, the server MUST close the Open as specified in section [3.3.4.17](#).

If **Lease.LeaseOpens** is empty, the server MUST NOT generate a Lease Break Notification. Instead, the server MUST complete the lease break call from the underlying object store with "NONE" as the new lease state, set **Lease.LeaseState** to "NONE", and take no further action.

If **Lease.LeaseOpens** is not empty, the server MUST construct a [Lease Break Notification \(section 2.2.23.2\)](#) message to send to the client. The server MUST set the **Command** in the SMB2 header to SMB2_OPLOCK_BREAK, the **MessageId** to 0xFFFFFFFFFFFFFFFF, and MUST set the **ProcessId** to 0. The server MUST set the **SessionId** and **TreeId** in the [SMB2 header](#) to 0. The **LeaseKey** field MUST be set to **Lease.LeaseKey**. **Lease.LeaseState** MUST be set to the new lease state indicated by the object store. The server MUST set **Open.OplockState** to Breaking for all opens in **Lease.LeaseOpens**. The server MUST set **Lease.Breaking** to TRUE, set **Lease.BreakToLeaseState** to the lease state indicated by the object store, and set **Lease.LeaseBreakTimeout** to the current time plus an implementation-specific default value in milliseconds. [<155>](#) The server MUST set the Flags field of the message to SMB2_NOTIFY_BREAK_LEASE_FLAG_ACK_REQUIRED, indicating to the client that lease acknowledgment is required.

The SMB2 **Lease Break Notification** is sent to the client using the connection specified in **Open.Connection** of the first Open in **Lease.LeaseOpens**. The message SHOULD NOT be signed. The server MUST start the oplock break acknowledgment timer as specified in [3.3.2.1](#). If there was an error in attempting to transmit the message to the client, the server MUST retry the send using the connection specified in **Open.Connection** of the next open in **Lease.LeaseOpens**. If the server fails to send transmit the message on any **Open.Connection** associated with this lease, the server MUST complete the lease break call from the underlying object store with "NONE" as the new lease state.

3.3.4.8 DFS Server Notifies SMB2 Server That DFS Is Active

In response to this event, the SMB2 server MUST set the global state variable **IsDfsCapable** to TRUE. If the DFS server is running on this computer, it MUST notify the SMB2 server that the DFS capability is available via this event.

3.3.4.9 DFS Server Notifies SMB2 Server That a Share Is a DFS Share

In response to this event, the SMB2 server MUST set the **Share.IsDfs** attribute of the **share** specified in section [3.3.1.6](#). When a DFS server running on this computer claims a share as a DFS share, it MUST notify the SMB2 server via this event.

3.3.4.10 DFS Server Notifies SMB2 Server That a Share Is Not a DFS Share

In response to this event, the SMB2 server MUST clear the **Share.IsDfs** attribute of the share specified in section [3.3.1.6](#).

3.3.4.11 Server Application Requests Security Context of the Client

An application running on the server issues a query for the security context of a client, specifying the **LocalOpen** to a named pipe that has been opened by the SMB2 server on behalf of the remote client.

The server MUST cycle through the entries in the **GlobalOpenTable** and locate the Open for which **Open.LocalOpen** matches the provided **LocalOpen**. If no Open is found, the request MUST be failed with STATUS_OBJECT_NAME_NOT_FOUND.

If **Open.Connection** is NULL, the request MUST be failed with STATUS_NO_TOKEN.

If **Open.TreeConnect.Share.Name** is not equal to "IPC\$" (indicating that the open is not a named pipe), the request SHOULD be failed with STATUS_ACCESS_DENIED.

If **Open.TreeConnect.Session.SecurityContext** is NULL, the request MUST be failed with STATUS_NO_TOKEN.

Otherwise, the server MUST return **Open.TreeConnect.Session.SecurityContext** to the caller.

3.3.4.12 Server Application Requests Closing a Session

The calling application provides **GlobalSessionId** of the session to be closed. The server MUST look up **Session** from the **GlobalSessionTable** where **Session.SessionGlobalId** is equal to **GlobalSessionId**, and remove it from the table. If there is no matching session, the call MUST return.

The server MUST deregister the session by invoking the event specified in [\[MS-SRVS\]](#) section 3.1.6.3, providing **GlobalSessionId** as the input parameter. **ServerStatistics.sts0_sopens** MUST be decreased by 1.

The server MUST close every **Open** in **Session.OpenTable** as specified in [3.3.4.17](#).

The server MUST disconnect every **TreeConnect** in **Session.TreeConnectTable** and deregister **TreeConnect** by invoking the event specified in [\[MS-SRVS\]](#) section 3.1.6.7, providing the tuple **<TreeConnect.Share.ServerName, TreeConnect.Share.Name>** and **TreeConnect.TreeGlobalId** as the input parameters. For each deregistered **TreeConnect**, **TreeConnect.Share.CurrentUses** MUST be decreased by 1.

The session MUST be torn down and freed.

3.3.4.13 Server Application Registers a Share

The calling application provides a share in **SHARE_INFO_503_I** structure as specified in [\[MS-SRVS\]](#) section 2.2.4.27 to register a share. The server MUST validate the **SHARE_INFO_503_I** structure as specified in [\[MS-SRVS\]](#) section 3.1.4.7. If any member in the structure is invalid, the server MUST return STATUS_INVALID_PARAMETER to the calling application. The server MUST look up the **Share** in the **ShareList**, where **shi503_servername** matches **Share.ServerName** and **shi503_netname** matches **Share.Name**. If a matching **Share** is found, the server MUST fail the call with an implementation-dependent error. Otherwise, the server MUST create a new Share with the following value set and insert it into **ShareList** and return STATUS_SUCCESS.

- **Share.Name** MUST be set to **shi503_netname**.
- **Share.Type** MUST be set to **shi503_type**.
- **Share.Remark** MUST be set to **shi503_remark**.
- **Share.LocalPath** MUST be set to **shi503_path**.
- **Share.ServerName** MUST be set to the **shi503_servername**.
- **Share.FileSecurity** MUST be set to **shi503_security_descriptor**.

- **Share.MaxUses** MUST be set to shi503_max_uses.
- **Share.CurrentUses** MUST be set to 0.
- **Share.CscFlags** MUST be set to 0.
- **Share.IsDfs** MUST be set to FALSE.
- **Share.DoAccessBasedDirectoryEnumeration** MUST be set to FALSE.
- **Share.AllowNamespaceCaching** MUST be set to FALSE.
- **Share.ForceSharedDelete** MUST be set to FALSE.
- **Share.RestrictExclusiveOpens** MUST be set to FALSE.
- **Share.ForceLevel2Oplock** MUST be set to FALSE.
- **Share.HashEnabled** MUST be set to FALSE.

If **Share.Name** is equal to IPC\$ or **Share.Type** does not have the STYPE_SPECIAL bit set, then **Share.ConnectSecurity** SHOULD be set to a **security descriptor** allowing all users. Otherwise, **Share.ConnectSecurity** SHOULD be set to a security descriptor allowing only administrators.

3.3.4.14 Server Application Updates a Share

The calling application provides a share in SHARE_INFO_503_I structure and SHARE_INFO_1005 structure as input parameters to update an existing **Share**. The server MUST validate the SHARE_INFO_503_I and SHARE_INFO_1005 structures as specified in [\[MS-SRVS\]](#) section 3.1.4.11. If any member in the structures is invalid, the server MUST return STATUS_INVALID_PARAMETER to the calling application. The server MUST look up the **Share** in the **ShareList**, where shi503_servername matches **Share.ServerName** and shi503_netname matches **Share.Name**. If the matching **Share** is found, the server MUST update the share with the following value set and return STATUS_SUCCESS to the calling application; otherwise the server MUST return an implementation-dependent error.

- **Share.FileSecurity** MUST be set to shi503_security_descriptor.
- **Share.Remark** MUST be set to shi503_remark.
- **Share.MaxUses** MUST be set to shi503_max_uses.
- **Share.CscFlags** MUST be set to the value of shi1005_flags masked by CSC_MASK as specified in [\[MS-SRVS\]](#) section 2.2.4.29.
- **Share.IsDfs** MUST be set to TRUE if shi1005_flags contains SHI1005_FLAGS_DFS or SHI1005_FLAGS_DFS_ROOT as specified in [\[MS-SRVS\]](#) section 2.2.4.29; otherwise, it MUST be set to FALSE.
- **Share.DoAccessBasedDirectoryEnumeration** MUST be set to TRUE if shi1005_flags contains SHI1005_FLAGS_ACCESS_BASED_DIRECTORY_ENUM bit as specified in [\[MS-SRVS\]](#) section 2.2.4.29; otherwise it MUST be set to FALSE.
- **Share.AllowNamespaceCaching** MUST be set to True if shi1005_flags contains SHI1005_FLAGS_ALLOW_NAMESPACE_CACHING bit as specified in [\[MS-SRVS\]](#) section 2.2.4.29; otherwise it MUST be set to FALSE.

- **Share.ForceSharedDelete** MUST be set to TRUE if shi1005_flags contains SHI1005_FLAGS_FORCE_SHARED_DELETE bit as specified in [\[MS-SRVS\]](#) section 2.2.4.29; otherwise it MUST be set to FALSE.
- **Share.RestrictExclusiveOpens** MUST be set to TRUE if shi1005_flags contains SHI1005_FLAGS_RESTRICT_EXCLUSIVE_OPENS bit as specified in [\[MS-SRVS\]](#) section 2.2.4.29; otherwise it MUST be set to FALSE.
- **Share.HashEnabled** MUST be set to TRUE if shi1005_flags contains SHI1005_FLAGS_ENABLE_HASH bit as specified in [\[MS-SRVS\]](#) section 2.2.4.29; otherwise it MUST be set to FALSE.
- **Share.ForceLevel2Oplock** MUST be set to TRUE if shi1005_flags contains SHI1005_FLAGS_FORCE_LEVELII_OPLOCK bit as specified in [\[MS-SRVS\]](#) section 2.2.4.29; otherwise, it MUST be set to FALSE.

3.3.4.15 Server Application Deregisters a Share

The calling application provides tuple <ServerName, ShareName> of the share that is being deregistered. The server MUST look up the **Share** in **ShareList** where **ServerName** matches **Share.ServerName** and **ShareName** matches **Share.Name**. If a **Share** is found, the server MUST remove it from the list and MUST return STATUS_SUCCESS to the calling application; otherwise, the server MUST return an implementation-specific error.

The server MUST close every **Open** in **GlobalOpenTable** where **Open.TreeConnect** is not NULL and **Open.TreeConnect.Share** matches the current share as specified in [3.3.4.17](#).

The server MUST enumerate every session in **GlobalSessionTable** and every tree connect in **Session.TreeConnectTable** to free all tree connect objects where **TreeConnect.Share** matches the current share.

3.3.4.16 Server Application Requests Querying a Share

The calling application provides tuple <ServerName, ShareName> of the share that is being queried. The server MUST look up the Share in **ShareList** where **ServerName** matches **Share.ServerName** and **ShareName** matches **Share.Name**. If the matching Share is found, the server MUST return a share in SHARE_INFO_503_I structure and SHARE_INFO_1005 structure with the following values set and return STATUS_SUCCESS to the calling application; otherwise the server MUST return an implementation-dependent error.

Output Parameters	SMB2 Share Properties
SHARE_INFO_503_I.shi503_netname	Share.Name
SHARE_INFO_503_I.shi503_type	Share.Type
SHARE_INFO_503_I.shi503_remark	Share.Remark
SHARE_INFO_503_I.shi503_permissions	0
SHARE_INFO_503_I.shi503_max_uses	Share.MaxUses
SHARE_INFO_503_I.shi503_current_uses	Share.CurrentUses
SHARE_INFO_503_I.shi503_path	Share.LocalPath
SHARE_INFO_503_I.shi503_passwd	Empty string

Output Parameters	SMB2 Share Properties
SHARE_INFO_503_I.shi503_servername	Share.ServerName
SHARE_INFO_503_I.shi503_security_descriptor	Share.FileSecurity
SHARE_INFO_1005.shi1005_flags	<p>ShareFlags MUST be set based on the individual share properties:</p> <ul style="list-style-type: none"> ▪ The server MUST set all flags contained in Share.CscFlags. ▪ The server MUST set the SHI1005_FLAGS_DFS bit if the per-share property Share.IsDfs is TRUE. ▪ The server MUST set the SHI1005_FLAGS_DFS_ROOT bit if the per-share property Share.IsDfs is TRUE. ▪ The server MUST set the SHI1005_FLAGS_ACCESS_BASED_DIRECTORY_ENUM bit if Share.DoAccessBasedDirectoryEnumeration is TRUE. ▪ The server MUST set the SHI1005_FLAGS_ALLOW_NAMESPACE_CACHING bit if Share.AllowNamespaceCaching is TRUE. ▪ The server MUST set the SHI1005_FLAGS_FORCE_SHARED_DELETE bit if Share.ForceSharedDelete is TRUE. ▪ The server MUST set the SHI1005_FLAGS_RESTRICT_EXCLUSIVE_OPENS bit if Share.RestrictExclusiveOpens is TRUE. ▪ The server MUST set the SHI1005_FLAGS_FORCE_LEVELII_OPLOCK bit if Share.ForceLevel2Oplock is TRUE. ▪ The server MUST set the SHI1005_FLAGS_ENABLE_HASH bit if Share.HashEnabled is TRUE.

3.3.4.17 Server Application Requests Closing an Open

The calling application provides **GlobalFileId** as input parameter. The server MUST look up **Open** in **GlobalOpenTable** where **Open.FileGlobalId** is equal to **GlobalFileId**, and, if the **Open** is found, the server MUST perform the following:

- Remove the **Open** from the **GlobalOpenTable**.
- If **Open.Connection** is not NULL, cancel all requests in **Open.Connection.RequestList** for which **Request.Open** matches the **Open**, as specified in section [3.3.5.16](#).
- Close the underlying **Open.LocalOpen**.
- If **Open.Session** is not NULL, remove the **Open** from **Open.Session.OpenTable**.

- If **Open.TreeConnect** is not NULL, decrease **Open.TreeConnect.OpenCount** by 1.
- If the server implements SMB 2.1 and supports leasing, and **Open.Lease** is not NULL:
 - The server MUST identify a **LeaseTable** by enumerating each entry in **GlobalLeaseTableList** to find the one whose **LeaseTable.LeaseList** contains **Open.Lease**.
 - The server MUST then remove the **Open** from **Lease.LeaseOpens**.
 - If **Lease.LeaseOpens** is now empty:
 - If **Lease.Breaking** is TRUE, the server MUST complete the lease break to the underlying object store with NONE as the new lease state. [<156>](#)
 - The server MUST remove the **Lease** from the **LeaseTable.LeaseList** and free the **Lease**.
 - If **LeaseTable.LeaseList** is now empty, the server MAY remove the **LeaseTable** from the **GlobalLeaseTableList** and free the **LeaseTable**.
- Provide **Open.FileGlobalId** as the input parameter and deregister the **Open** by invoking the event specified in [\[MS-SRVS\]](#) section 3.1.6.5.
- The **Open** object is then freed.
- Return STATUS_SUCCESS to the calling application.

If no **Open** is found, the call MUST return an implementation-dependent error.

3.3.4.18 Server Application Queries a Session

The calling application provides **GlobalSessionId** as an identifier for the **Session**. The server MUST look up session in **GlobalSessionTable** where **GlobalSessionId** is equal to **Session.SessionGlobalId**. If **Session** is found, the server MUST return a session in SESSION_INFO_502 structure as specified in [\[MS-SRVS\]](#) section 2.2.4.15 with the following values set and return STATUS_SUCCESS to the calling application.

SESSION_INFO_502 Parameters	SMB2 Session Properties
sesi502_cname	Session.Connection.ClientName
sesi502_username	Session.UserName
sesi502_num_opens	The count of entries in Session.OpenTable
sesi502_time	The current time minus Session.CreationTime , in seconds
sesi502_idle_time	The current time minus Session.IdleTime , in seconds
sesi502_user_flags	SESS_GUEST if the security principal associated with Session.SecurityContext is a guest user (see section 3.3.5.5.3); otherwise zero
sesi502_cltype_name	Empty string
sesi502_transport	Session.Connection.TransportName

If no **Session** is found, the server MUST return an implementation-dependent error.

3.3.4.19 Server Application Queries a TreeConnect

The calling application provides **GlobalTreeConnectId** as an identifier for the tree connect. The server MUST enumerate all session entries in **GlobalSessionTable** and look up all **TreeConnect** entries in **Session.TreeConnectTable** where **GlobalTreeConnectId** is equal to **TreeConnect.TreeGlobalId**. If **TreeConnect** is found, the server MUST return **ServerName** and a **CONNECT_INFO_1** structure as specified in [\[MS-SRVS\]](#) section 2.2.4.2 with the following values set and return **STATUS_SUCCESS** to the calling application.

Output Parameters	SMB2 TreeConnect Properties
con1_id	TreeConnect.TreeGlobalId
con1_type	TreeConnect.Share.Type
con1_num_opens	TreeConnect.OpenCount
con1_num_users	1
con1_time	Current time minus Treeconnect.CreationTime , in seconds.
con1_username	TreeConnect.Session.UserName
con1_netname	TreeConnect.Share.Name
ServerName	TreeConnect.Share.ServerName

If no **TreeConnect** is found, the server MUST return an implementation-dependent error.

3.3.4.20 Server Application Queries an Open

The calling application provides **GlobalFileId** as an identifier for the **Open**. The server MUST look up open in **GlobalOpenTable** where **GlobalFileId** is equal to **Open.FileGlobalId**. If **Open** is found, the server MUST return an open in **FILE_INFO_3** structure as specified in [\[MS-SRVS\]](#) section 2.2.4.7 with the following values set and return **STATUS_SUCCESS** to the calling application.

FILE_INFO_3 Parameters	SMB2 Open Properties
fi3_id	Open.FileGlobalId
fi3_permissions	Open.GrantedAccess
fi3_num_locks	Open.LockCount
fi3_path_name	Open.PathName
fi3_username	Open.Session.UserName , or empty if Open.Session is NULL

If no **Open** is found, the server MUST return an implementation-dependent error.

3.3.4.21 Server Application Requests Transport Binding Change

The application provides:

- **TransportName**: A string containing an implementation-specific name of the transport.

- **ServerName**: An optional string containing the name of the server to be used for binding the transport.
- **EnableFlag**: A Boolean flag indicating whether to enable or disable the transport.

The server MUST use implementation-specific [<157>](#) means to determine whether **TransportName** is an eligible transport entry as specified in section [2.1](#), and if not, the server MUST return `ERROR_NOT_SUPPORTED` to the caller.

If **EnableFlag** is TRUE, the server SHOULD obtain binding information for the transport from the appropriate standards assignments as specified in section [1.9](#) and **ServerName** [<158>](#) and MUST attempt to start listening on the requested transport endpoint.

If **EnableFlag** is FALSE, the server MUST attempt to stop listening on the transport indicated by **TransportName**.

If the attempt to start or stop listening on the transport succeeds, the server MUST return `STATUS_SUCCESS` to the caller. Otherwise, it MUST return an implementation-dependent error.

3.3.4.22 Server Application Enables the SMB2 Server

The server MUST verify that the caller of this interface is the server service [\[MS-SRVS\]](#), in an implementation-specific manner. In this case only, the **ServerEnabled** ADM element MUST be set to TRUE.

3.3.4.23 Server Application Disables the SMB2 Server

The server MUST verify, in an implementation-specific manner, that the caller of this interface is the server service [\[MS-SRVS\]](#). Only if so, the server MUST take the following actions:

- The server MUST set **ServerEnabled** to FALSE to prevent accepting new connections.
- For each session in **GlobalSessionTable**, the server MUST take the following actions:
 - The server MUST disconnect **Session.Connection**.
 - The server MUST close the session as specified in section [3.3.4.12](#), providing **Session.SessionGlobalId** as the input parameter.
- For each **Open** in **GlobalOpenTable**, the server MUST close the open as specified in section [3.3.4.17](#), providing **Open.FileGlobalId** as the input parameter.
- The server MUST remove and free all the shares in **ShareList**.
- For each connection in **ConnectionList**, the server MUST invoke the event specified in [\[MS-SRVS\]](#) section 3.1.6.16 to update the connection count by providing the tuple `<Connection.TransportName,FALSE>`. The server MUST remove and free all connections in **ConnectionList**. The server MUST remove and free all connections in **ConnectionList**.

3.3.4.24 Server Application Requests Server Statistics

The server MUST return the **ServerStatistics** in a **STAT_SERVER_0** structure as specified in [\[MS-SRVS\]](#) section 2.2.4.39 to the server application with the following values:

STAT_SERVER_0 members	SMB2 ServerStatistics Properties
sts0_start	zero
sts0_fopens	ServerStatistics.sts0_fopens
sts0_devopens	zero
sts0_jobsqueued	ServerStatistics.sts0_jobsqueued
sts0_sopens	ServerStatistics.sts0_sopens
sts0_stimedout	ServerStatistics.sts0_stimedout
sts0_serrorout	zero
sts0_pwerrors	ServerStatistics.sts0_pwerrors
sts0_permerrors	ServerStatistics.sts0_permerrors
sts0_syserrors	zero
sts0_bytessent_low	ServerStatistics.sts0_bytessent_low
sts0_bytessent_high	ServerStatistics.sts0_bytessent_high
sts0_bytesrcvd_low	ServerStatistics.sts0_bytesrcvd_low
sts0_bytesrcvd_high	ServerStatistics.sts0_bytesrcvd_high
sts0_avresponse	zero
sts0_reqbufneed	zero
sts0_bigbufneed	zero

3.3.5 Processing Events and Sequencing Rules

The SMB 2 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

If **ServerEnabled** is FALSE, the server MUST NOT accept any incoming connections. Otherwise, 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.CommandSequenceWindow is set to a sequence window, as specified in section [3.3.1.1](#), 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.NegotiateDialect is set to 0xFFFF.

Connection.Dialect is set to "Unknown".

Connection.ShouldSign is set to FALSE.

Connection.ClientName is set to be a null-terminated Unicode string of an IP address if the connection is on TCP port 445, or a NetBIOS host name if the connection is on TCP port 139.

Connection.MaxTransactSize is set to 0.

Connection.SupportsMultiCredit is set to FALSE.

Connection.TransportName is set to the implementation-specific name of the transport used by this connection [<159>](#159) as obtained by implementation-specific means from the transport that indicated the incoming connection.

The server MUST invoke the event specified in [\[MS-SRVS\]](#MS-SRVS) section 3.1.6.16 to update the connection count by providing the tuple **<Connection.TransportName,TRUE>**.

This connection MUST be inserted into the global **ConnectionList**.

3.3.5.2 Receiving Any Message

If the received request is not an SMB2 CANCEL, the server MUST create a new **Request** object initialized as follows, and insert it into the **Connection.RequestList** before verifying the connection state, sequence number, or signature.

- **Request.MessageId** MUST be set to the **MessageId** value in the SMB2 header.
- **Request.AsyncId** MUST be set to 0.
- **Request.CancelRequestId** MUST be set to a unique identifier generated by the server. In each invocation of an object store operation, the server MUST pass the **CancelRequestId** as an additional parameter to the operation, in order to support cancellation of in-progress operations as specified in section [<160>](#3.3.5.16.160)
- **Request.Open** MUST be set to NULL.

For a compound request, the server MUST register each SMB2 command as a separate entry in the **Connection.RequestList**, and **Request.MessageId** MUST be set to the **MessageId** values from the individual command headers.

For every message received, the server MUST calculate the total number of bytes in the message and update the values of **ServerStatistics.sts0_bytesrcvd_low** and **ServerStatistics.sts0_bytesrcvd_high**.

3.3.5.2.1 Verifying the Connection State

If the request being received is not an [SMB2 NEGOTIATE Request](#SMB2_NEGOTIATE_Request) or a traditional SMB_COM_NEGOTIATE, as described in section [1.7](#1.7), and **Connection.NegotiateDialect** is 0xFFFF or 0x02FF, the server MUST disconnect the connection, as specified in section [3.3.7.1](#3.3.7.1), and send no reply.

3.3.5.2.2 Verifying the Sequence Number

If the received request is an SMB2 CANCEL, this section MUST be skipped.

If the received request is an SMB_COM_NEGOTIATE, as described in section [1.7](#), the server MUST assume that **MessageId** is zero for this request.

The server MUST check that the **MessageId** for the received request falls within the **Connection.CommandSequenceWindow**, as specified in section [3.3.1.7](#).

If **Connection.SupportsMultiCredit** is TRUE and the **CreditCharge** field in the SMB2 header is greater than zero, the server MUST check that a number of **CreditCharge** consecutive sequence numbers starting from **MessageId** fall within the **Connection.CommandSequenceWindow**.

If the server determines that the **MessageId** or the range of **MessageIds** for the incoming request is not valid, the server SHOULD fail the request, as specified in section [3.3.4.4](#), with the error code STATUS_INVALID_PARAMETER, [<161>](#) and MUST disconnect the connection, as specified in section [3.3.7.1](#). Otherwise, the server MUST remove the **MessageId** or the range of **MessageIds** from the **Connection.CommandSequenceWindow**.

3.3.5.2.3 Verifying the Signature

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 **GlobalSessionTable** 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 \(section 3.3.4.4\)](#), 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 disconnect the connection as specified in section [3.3.7.1](#). If signature verification succeeds, the server MUST continue processing on the packet. [<162>](#)

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 **GlobalSessionTable** 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 [<163>](#) also disconnect the connection, as specified in section [3.3.7.1](#). If either the session is not found, or **Session.ShouldSign** is FALSE, the server continues processing on the packet.

If the connection is disconnected, the server MUST remove the connection from the **ConnectionList**, as specified in section [3.3.7.1](#).

3.3.5.2.4 Verifying the Credit Charge and the Payload Size

If **Connection.SupportsMultiCredit** is TRUE, the server MUST verify the **CreditCharge** field in the SMB2 header and the payload size of the request.

- If **CreditCharge** is zero and the payload size of the request is greater than 64 kilobytes, the server MUST fail the request with the error code STATUS_INVALID_PARAMETER.
- If **CreditCharge** is greater than zero, the server MUST calculate the expected **CreditCharge** for the current operation using the formula specified in section [3.2.4.1.5](#). If the calculated credit number is greater than the **CreditCharge**, the server MUST fail the request with the error code STATUS_INVALID_PARAMETER.

If **Connection.SupportsMultiCredit** is FALSE and the payload size of the request is greater than 64 kilobytes, the server SHOULD [<164>](#) fail the request with the error code STATUS_INVALID_PARAMETER.

3.3.5.2.5 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 3.3.4.4, with the error code STATUS_INVALID_PARAMETER. The server MAY <165> also disconnect the connection.

The server MUST disconnect, as specified in section 3.3.7.1, 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.

3.3.5.2.6 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 <166> fail requests in a compound chain that try to go async. There are two different styles of compounded requests, which are described in the following sections.

The two styles MUST NOT be intermixed in the same transport send, and in such a case, the server SHOULD <167> fail each of the requests with STATUS_INVALID_PARAMETER.

3.3.5.2.6.1 Handling Compounded Unrelated Requests

If SMB2_FLAGS_RELATED_OPERATIONS is off in the **Flags** field of the [SMB2 header](#) of every request, the received requests 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.

3.3.5.2.6.2 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 received requests MUST be handled as a series of compounded related requests. If the first request has SMB2_FLAGS_RELATED_OPERATIONS set, the server SHOULD <168> 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 used for **FileId**, **SessionId**, and **TreeId** MUST be the ones used in processing the previous request or generated for the previous resulting response. When all operations are complete, the responses SHOULD <169> be compounded into a single response to return to the client.

If the responses are compounded, the server SHOULD set SMB2_FLAGS_RELATED_OPERATIONS in the **Flags** field of the SMB2 header of all responses except the first one. This indicates that the response was part of a compounded chain.

3.3.5.2.7 Updating Idle Time

For every request received, the server MUST locate the session, using the **SessionId** in the SMB2 header of the request to do a lookup on the **GlobalSessionTable**. If a session is found, **Session.IdleTime** MUST be set to the current time. If the request does not have an SMB2 header following the syntax specified in section [2.2.1](#) or no session is found, no action regarding the idle time is taken.

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 described in section [1.7](#).

This request is defined in [\[MS-SMB\]](#) section 2.2.4.5.1, with the SMB header defined in section [2.2.3.1](#). If the request matches the format described there, and **Connection.NegotiateDialect** is 0xFFFF, processing MUST continue as specified in [3.3.5.3.1](#). Otherwise, the server MUST disconnect the connection, as specified in section [3.3.7.1](#), without sending a response.

3.3.5.3.1 SMB 2.1 Support

If the server does not support SMB 2.1, processing MUST continue as specified in [3.3.5.3.2](#).

If the server supports SMB 2.1, the server MUST scan the dialects provided for the dialect string "SMB 2.???". If the string is not present, continue to section 3.3.5.3.2. If the string is present, the server MUST respond with an SMB2 NEGOTIATE Response as specified in [2.2.4](#). If the string is present and underlying TCP endpoint is 445, **Connection.SupportsMultiCredit** MUST be set to TRUE.

The server MUST set the command of the SMB2 header to SMB2 NEGOTIATE and MUST set the **ProcessId** to 0. 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 [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 0x02FF.
- **ServerGuid** is set to the global **ServerGuid** value.
- The **Capabilities** field MUST be set to a combination of zero or more of the following bit values, as specified in section [2.2.4](#):
 - SMB2_GLOBAL_CAP_DFS if the server supports the Distributed File System.
 - SMB2_GLOBAL_CAP_LEASING if the server supports leasing.
 - SMB2_GLOBAL_CAP_LARGE_MTU if **Connection.SupportsMultiCredit** is TRUE.
- **MaxTransactSize** is set to the maximum buffer size [<170>](#), 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. **Connection.MaxTransactSize** MUST be set to **MaxTransactSize**.

- **MaxReadSize** is set to the maximum size, in bytes, of the Length in an SMB2 READ Request ([2.2.19](#)) that the server will accept on the transport that established this connection. [<171>](#)
- **MaxWriteSize** is set to the maximum size, in bytes, of the Length in an SMB2 Write Request ([2.2.21](#)) that the server will accept on the transport that established this connection. [<172>](#)
- **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 data being returned in the **Buffer** field.
- **Buffer** is filled with a GSS token, generated as follows. Alternatively, an empty **Buffer** MAY be returned, which will elicit client-initiated authentication with an authentication protocol of the client's choice.

The generation of the GSS token for the SMB2 NEGOTIATE Response MUST be done as specified in [\[MS-SPNG\] 3.2.5.2](#). 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 3.2.5.2](#).

Connection.NegotiateDialect MUST be set to 0x02FF, and the response is sent to the client.

3.3.5.3.2 SMB 2.002 Support

The server MUST scan the dialects provided for the dialect string "SMB 2.002". [<173>](#) 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, as specified in [section 3.3.7.1](#), without sending a response.

The server MUST set the command of the [SMB2 header](#) to SMB2 NEGOTIATE and MUST set the **ProcessId** to 0. 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. [<174>](#)
- **ServerGuid** is set to the global **ServerGuid** value.
- If the server supports the Distributed File System, set the SMB2_GLOBAL_CAP_DFS bit in the **Capabilities** field of the negotiate response.
- **MaxTransactSize** is set to the maximum buffer size [<175>](#), 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. **Connection.MaxTransactSize** MUST be set to **MaxTransactSize**.

- **MaxReadSize** is set to the maximum size, in bytes, of the Length in an [SMB2_READ Request \(2.2.19\)](#) 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_WRITE Request \(2.2.21\)](#) 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 data being returned in the **Buffer** field.
- **Buffer** is filled with a GSS token, generated as follows. Alternatively, an empty **Buffer** MAY be returned, which will elicit client-initiated authentication with an authentication protocol of the client's choice.

The generation of the GSS token for the SMB2 NEGOTIATE Response MUST be done as specified in [\[MS-SPNG\]](#) section 3.2.5.2. 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 3.2.5.2.

Connection.Dialect MUST be set to "2.002", **Connection.NegotiateDialect** MUST be set to 0x0202, and the response is sent to the client.

Connection.SupportsMultiCredit MUST be set to FALSE.

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.NegotiateDialect** is 0x0202 or 0x0210, the server MUST disconnect the connection, as specified in section [3.3.7.1](#), and not reply.

If the server implements SMB2.1, the server MUST set **Connection.ClientGuid** to the **ClientGuid** field of the [SMB2 Negotiate request](#).

If SMB2_NEGOTIATE_SIGNING_REQUIRED is set in **SecurityMode**, the server MUST set **Connection.ShouldSign** to TRUE.

The server MUST select the greatest common dialect between the dialects it supports and the Dialects array of the SMB2 NEGOTIATE request. If a common dialect is not found, the server MUST fail the request with STATUS_NOT_SUPPORTED.

If a common dialect is found, the server MUST set **Connection.Dialect** to "2.002" or "2.100" and **Connection.NegotiateDialect** to 0x0202 or 0x0210 accordingly, to reflect the dialect selected. The server MUST then construct an [SMB2 NEGOTIATE Response](#), as specified in section [2.2.4](#), with the following specific values, and return STATUS_SUCCESS to the client.

If the common dialect is SMB 2.1 and the underlying TCP endpoint is on port 445, **Connection.SupportsMultiCredit** MUST be set to TRUE; otherwise, it MUST be set to FALSE.

- **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 the common dialect.
- **ServerGuid** is set to the global **ServerGuid** value.
- The **Capabilities** field MUST be set to a combination of zero or more of the following bit values, as specified in section 2.2.4:
 - SMB2_GLOBAL_CAP_DFS if the server supports the Distributed File System.
 - SMB2_GLOBAL_CAP_LEASING if the server supports leasing.
 - SMB2_GLOBAL_CAP_LARGE_MTU if **Connection.SupportsMultiCredit** is TRUE.
- **MaxTransactSize** is set to the maximum buffer size [<176>](#), 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. **Connection.MaxTransactSize** MUST be set to **MaxTransactSize**.
- **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 WRITE Request \(section 2.2.21\)](#) 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 data being returned in the **Buffer** field.
- **Buffer** is filled with the GSS token, generated as follows. Alternatively, an empty **Buffer** MAY be returned, which will elicit client-initiated authentication with an authentication protocol of the client's choice.

The generation of the GSS token for the SMB2 NEGOTIATE Response MUST be done as specified in [\[MS-SPNG\]](#) section 3.2.5.2. 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 3.2.5.2.

The status code returned by this operation MUST be one of those defined in [\[MS-ERREF\]](#). Common status codes returned by this operation include:

- STATUS_SUCCESS
- STATUS_INSUFFICIENT_RESOURCES

- STATUS_INVALID_PARAMETER
- STATUS_NOT_SUPPORTED

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 **GlobalSessionTable** 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 and the server does not implement SMB 2.1, the server MUST fail the session setup request with STATUS_REQUEST_NOT_ACCEPTED. Otherwise, proceed to step 6.
6. If **Session.State** is Valid and the server does implement SMB 2.1, the server MUST process the session setup request as specified in section [3.3.5.5.2](#). Otherwise, proceed to step 7.
7. The server MUST continue processing the request as specified in section [3.3.5.5.3](#).

The status code returned by this operation MUST be one of those defined in [\[MS-ERREF\]](#). Common status codes returned by this operation include:

- STATUS_LOGON_FAILURE
- STATUS_INSUFFICIENT_RESOURCES
- STATUS_SUCCESS
- STATUS_MORE_PROCESSING_REQUIRED
- STATUS_INVALID_PARAMETER
- STATUS_USER_SESSION_DELETED
- STATUS_REQUEST_NOT_ACCEPTED
- STATUS_PASSWORD_EXPIRED
- SEC_E_INVALID_TOKEN
- SEC_E_NO_CREDENTIALS

3.3.5.5.1 Authenticating a New Session

A session object MUST be allocated for this request. The session MUST be inserted into the **GlobalSessionTable** and a unique **Session.SessionId** is assigned to serve as a lookup key in the table. The server MUST register the session by invoking the event specified in [\[MS-SRVS\]](#) section 3.1.6.2 and assign the return value to **Session.SessionGlobalId**. **ServerStatistics.sts0_sopens**

MUST be increased by 1. The SMB2 server MUST reserve -1 as an invalid **SessionId** and 0 as a **SessionId** for which no session exists. 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 NULL, indicating that it is uninitialized.
- **Session.ShouldSign** is set to FALSE.
- **Session.OpenTable** is set to an empty table.
- **Session.TreeConnectTable** is set to an empty table.
- **Session.IsAnonymous** is set to FALSE.
- **Session.CreationTime** is set to the current time.
- **Session.IdleTime** is set to the current time.

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](#). Note that the existing **Session.SessionKey** will be retained.

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 SHOULD use the configured authentication protocol to obtain the next GSS output token for the authentication exchange. [<177>](#)

If the authentication protocol indicates an error, the server MUST fail the session setup request with the error received by placing the 32-bit NTSTATUS code received into the **Status** field of the SMB2 header. The server MUST remove the session object from **GlobalSessionTable** and deregister the session by invoking the event specified in [\[MS-SRVS\]](#) section 3.1.6.3, providing **Session.SessionGlobalId** as an input parameter. **ServerStatistics.sts0_sopens** MUST be decreased by 1. The server MUST close every **Open** in **Session.OpenTable** as specified in section [3.3.4.17](#). The server MUST deregister every **TreeConnect** in **Session.TreeConnectTable** by providing the tuple **<TreeConnect.Share.ServerName, TreeConnect.Share.Name>** and **TreeConnect.TreeGlobalId** as the input parameters and invoking the event specified in [\[MS-SRVS\]](#) section 3.1.6.7. For each deregistered **TreeConnect**, **TreeConnect.Share.CurrentUses** MUST be decreased by 1. All the tree connects in **Session.TreeConnectTable** MUST be removed and freed. The session object MUST also be freed, and the error response MUST be sent to the client. **ServerStatistics.sts0_perrors** MUST be increased by 1.

The following errors can be returned by the GSS-API interface as specified in [\[RFC2743\]](#). STATUS_PASSWORD_EXPIRED SHOULD be treated as GSS_S_CREDENTIALS_EXPIRED, SEC_E_INVALID_TOKEN SHOULD be treated as GSS_S_DEFECTIVE_TOKEN, and SEC_E_NO_CREDENTIALS SHOULD be treated as GSS_S_NO_CRED. All other errors SHOULD be

treated as a GSS_S_FAILURE error code. 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
- STATUS_LOGON_FAILURE

If the authentication protocol 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** indicates the length of the output token, and **SecurityBufferOffset** indicates 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:

1. The status code in the SMB2 header of the response MUST be set to STATUS_SUCCESS.
2. If **Connection.ClientCapabilities** is 0, the server MUST set **Connection.ClientCapabilities** to the capabilities received in the [SMB2 SESSION SETUP Request](#).
3. 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. The server MUST invoke the GSS_Inquire_context call as specified in [\[RFC2743\]](#) section 2.2.6, passing the **Session.SecurityContext** as the input parameter, and set **Session.UserName** to the returned "src_name".
4. The server MUST invoke the GSS_Inquire_context call as specified in [\[RFC2743\]](#) section 2.2.6, passing the **Session.SecurityContext** as the context_handle parameter.

If the returned anon_state is TRUE, the server MUST set **Session.IsAnonymous** to TRUE and set the SMB2_SESSION_FLAG_IS_NULL flag in the **SessionFlags** field of the SMB2 SESSION_SETUP Response.

Otherwise, if the returned src_name corresponds to an implementation-specific guest user, [<178>](#) the server MUST set the SMB2_SESSION_FLAG_IS_GUEST in the **SessionFlags** field of the SMB2 SESSION_SETUP Response.

5. 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, if **Connection.ShouldSign** is set to TRUE, OR the global [RequireMessageSigning](#) is set to TRUE, **Session.ShouldSign** MUST be set to TRUE.
6. If **Session.ShouldSign** is TRUE, the server MUST sign the final session setup response before sending it to the client.
7. If **Session.SessionKey** is NULL, the server MUST query the session key for this authentication from the underlying authentication protocol and store the session key in **Session.SessionKey**. **Session.SessionKey** MUST be set as described in section [3.3.1.8](#), using the value queried from the GSS protocol. For how this value is calculated for Kerberos authentication via GSS-API, see [\[MS-KILE\]](#) section 3.1.1.2. For how this value is calculated for NTLM authentication via GSS-API, see [\[MS-NLMP\]](#) section 3.1.5.1.
8. If the **PreviousSessionId** field of the request is not equal to zero, the server MUST take the following actions:
 1. The server MUST look up the old session based on this value. If no session is found, no other processing is necessary.
 2. 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.
 1. If the server determines the authentications were for the same user, the server MUST remove the old session from the **GlobalSessionTable** and deregister the session by invoking the event specified in [\[MS-SRVS\]](#) section 3.1.6.3, providing **Session.SessionGlobalId** as an input parameter. **ServerStatistics.sts0_sopens** MUST be decreased by 1. The server MUST close every **Open** in **Session.OpenTable** of the old session as specified in section [3.3.4.17](#). Any tree connects in **Session.TreeConnectTable** of the old session MUST be deregistered by invoking the event specified in [\[MS-SRVS\]](#) section 3.1.6.7, providing the tuple **<TreeConnect.Share.ServerName, TreeConnect.Share.Name>** and **Treeconnect.TreeGlobalId** as input parameters, and each of them MUST be freed. For each deregistered **TreeConnect**, **TreeConnect.Share.CurrentUses** MUST be decreased by 1.
 2. If the server determines that the authentications were for different users, the server MUST ignore the **PreviousSessionId** value.
9. **Session.State** MUST be set to Valid.
10. **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 3.3.1. If the GSS mechanism indicates that this is not the final message of the authentication exchange, the following additional step 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 **GlobalSessionTable**. 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.

Otherwise, the server MUST remove this session from the **GlobalSessionTable**, and deregister the session by invoking the event specified in [\[MS-SRVS\]](#) section 3.1.6.3, providing **Session.SessionGlobalId** as input parameter. **ServerStatistics.sts0_sopens** MUST be decreased by 1. The server MUST close every **Open** in **Session.OpenTable** of the old session as specified in section [3.3.4.17](#). Any tree connects in **Session.TreeConnectTable** of the old session MUST be deregistered by invoking the event specified in [\[MS-SRVS\]](#) section 3.1.6.7, providing the tuple **<TreeConnect.Share.ServerName, TreeConnect.Share.Name>** and **Treeconnect.TreeGlobalId** as input parameters, and each of them MUST be freed. For each deregistered **TreeConnect**, **TreeConnect.Share.CurrentUses** MUST be decreased by 1. The server MUST construct an [SMB2 LOGOFF Response](#) with a status code of STATUS_SUCCESS, following the syntax specified in section [2.2.8](#), and send it to the client. The session itself is then freed.

The status code returned by this operation MUST be one of those defined in [\[MS-ERREF\]](#). Common status codes returned by this operation include:

- STATUS_SUCCESS
- STATUS_USER_SESSION_DELETED
- STATUS_INVALID_PARAMETER
- STATUS_NETWORK_SESSION_EXPIRED
- STATUS_ACCESS_DENIED

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 **GlobalSessionTable** 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 provide <server name, share name> parsed from the request message to invoke the event specified in [\[MS-SRVS\]](#) section 3.1.6.8, to normalize the server name by resolving server aliases and evaluating share scope. The server MUST use <normalized server name, share name> to look up the **Share** in **ShareList**. If no share with a matching share name and server name 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. The server MUST provide the tuple **<server name, share name>** to invoke the event specified in [\[MS-SRVS\]](#) section 3.1.6.15 to get the total number of current uses of the share. If the current uses is equal to or greater than **Share.MaxUses**, the server MUST fail the request with STATUS_REQUEST_NOT_ACCEPTED. Otherwise, the server MUST allocate a tree connect object and insert it into **Session.TreeConnectTable**. The server MUST provide the tuple **<server name, share name>** and MUST register **TreeConnect** by invoking the event specified in [\[MS-SRVS\]](#) section 3.1.6.6 and assign the return value to **TreeConnect.TreeGlobalId**. The other initial values MUST be set as follows:

- **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.
- **TreeConnect.OpenCount** MUST be set to 0.
- **TreeConnect.CreationTime** MUST be set to current time.
- **TreeConnect.Share.CurrentUses** MUST be increased by 1.

The tree connect response MUST be constructed following the syntax specified in section [2.2.10](#), as described here:

- **ShareFlags** MUST be set based on the individual share properties (**Share.CscFlags**, **Share.DoAccessBasedDirectoryEnumeration**, **Share.AllowNamespaceCaching**, **Share.ForceSharedDelete**, **Share.RestrictExclusiveOpens**, **Share.HashEnabled**, **Share.ForceLevel2Oplock**, **Share.IsDfs**.)
 - The server MUST set all flags contained in **Share.CscFlags**.
 - The server SHOULD [<179>](#) set the SHI1005_FLAGS_DFS bit if the per-share property **Share.IsDfs** is TRUE, indicating that the share is part of a DFS namespace.
 - The server SHOULD [<180>](#) set the SHI1005_FLAGS_DFS_ROOT bit if the per-share property **Share.IsDfs** is TRUE, indicating that the share is part of a DFS namespace.
 - The server MUST set the SHI1005_FLAGS_ACCESS_BASED_DIRECTORY_ENUM bit if **Share.DoAccessBasedDirectoryEnumeration** is TRUE.
 - The server MUST set the SHI1005_FLAGS_ALLOW_NAMESPACE_CACHING bit if **Share.AllowNamespaceCaching** is TRUE.
 - The server MUST set the SHI1005_FLAGS_FORCE_SHARED_DELETE bit if **Share.ForceSharedDelete** is TRUE.
 - The server MUST set the SHI1005_FLAGS_RESTRICT_EXCLUSIVE_OPENS bit if **Share.RestrictExclusiveOpens** is TRUE.
 - The server MUST set the SHI1005_FLAGS_ENABLE_HASH bit if **Share.HashEnabled** is TRUE.
 - The server MUST set the SHI1005_FLAGS_FORCE_LEVELII_OPLOCK bit if **Share.ForceLevel2Oplock** is TRUE.

- **ShareType** MUST be set based on the resource being shared, as indicated by **Share.Type**:
 - If this share provides access to named pipes, as indicated by resource type STYPE_IPC, **ShareType** MUST be set to SMB2_SHARE_TYPE_PIPE.
 - If this share provides access to a printer, as indicated by the resource type STYPE_PRINTQ, **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.

The status code returned by this operation MUST be one of those defined in [\[MS-ERREF\]](#). Common status codes returned by this operation include:

- STATUS_SUCCESS
- STATUS_ACCESS_DENIED
- STATUS_INSUFFICIENT_RESOURCES
- STATUS_BAD_NETWORK_NAME
- STATUS_INVALID_PARAMETER
- STATUS_USER_SESSION_DELETED
- STATUS_NETWORK_SESSION_EXPIRED

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 **GlobalSessionTable**. 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.

For any **Open** in **Session.OpenTable**, if **Open.TreeConnect** matches the tree connect being disconnected, the server MUST close the **Open** as specified in section [3.3.4.17](#).

The server MUST provide the tuple **<TreeConnect.Share.ServerName, TreeConnect.Share.Name>** and **TreeConnect.TreeGlobalId** as input parameters and deregister **TreeConnect** by invoking the event specified in [\[MS-SRVS\]](#) section 3.1.6.7. **TreeConnect.Share.CurrentUses** MUST be decreased by 1. 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.

The status code returned by this operation MUST be one of those defined in [\[MS-ERREF\]](#). Common status codes returned by this operation include:

- STATUS_SUCCESS
- STATUS_INSUFFICIENT_RESOURCES
- STATUS_USER_SESSION_DELETED
- STATUS_NETWORK_NAME_DELETED
- STATUS_INVALID_PARAMETER
- STATUS_NETWORK_SESSION_EXPIRED
- STATUS_ACCESS_DENIED

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 **GlobalSessionTable**. 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 verify the request size. If the size of the SMB2 CREATE Request (excluding the SMB2 header) is less than specified in the **StructureSize** field, then the request MUST be failed with STATUS_INVALID_PARAMETER.

The server MUST extract the target path name for the create from the [SMB2 CREATE Request](#).

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 verify the value of **IsDfsCapable**:

- If **IsDfsCapable** is TRUE, the server MUST invoke the interface defined in [\[MS-DFSC\]](#) section 3.2.4.1 to normalize the path name by supplying the target path name.
- If **IsDfsCapable** is FALSE, the server MUST fail the request with STATUS_FS_DRIVER_REQUIRED.

If the request received does not have the SMB2_FLAGS_DFS_OPERATIONS flag set in the **Flags** field of the SMB2 header, or **TreeConnect.Share.IsDfs** is FALSE, the server MUST NOT invoke normalization and continue the create process.

If normalization fails, the server MUST fail the create request with the error code returned by the DFS normalization routine.

If the normalization procedure succeeds, returning an altered target name, the modified name MUST be used for further operations.

If the file name length is greater than zero and the first character is a separator character, the server MUST fail the request with STATUS_OBJECT_NAME_INVALID. If the file name fails to conform with the specification of a relative pathname in [\[MS-FSCC\]](#) section 2.1.5, the server MUST fail the request with STATUS_OBJECT_NAME_INVALID.

If any flags other than those defined in [\[MS-FSCC\]](#) section 2.6 are set in the **FileAttributes** field of the request packet, the server MUST fail the request with STATUS_INVALID_PARAMETER.

If any of the bits in the mask 0x0CE0FE00 are set in the **DesiredAccess** field, the server SHOULD [<181>](#) fail the create request with STATUS_ACCESS_DENIED.

If the share that is the target of the create request is the IPC\$ share and **Session.IsAnonymous** is TRUE, the server MUST invoke the event specified in [\[MS-SRVS\]](#) section 3.1.6.17 by providing the target name as the input parameter. If the event returns FALSE, indicating that no matching named pipe is found that allows an anonymous user, the server MUST fail the request with STATUS_ACCESS_DENIED and increase **Server.Statistics.sts0_permerrors** by 1. Otherwise, the server MUST continue the open processing.

If the share that is the target of the create request is a printer, the server MUST validate the **DesiredAccess** and **CreateDisposition** fields of the request. If the **DesiredAccess** value does not include one or more of the FILE_WRITE_DATA, FILE_APPEND_DATA, or GENERIC_WRITE bits, the server SHOULD [<182>](#) fail the request with STATUS_NOT_SUPPORTED. If the **DesiredAccess** value contains any other bits, the server MUST fail the request with STATUS_NOT_SUPPORTED. If the **CreateDisposition** value is other than FILE_CREATE, the server SHOULD [<183>](#) fail the request with STATUS_OBJECT_NAME_NOT_FOUND.

If any intermediate component of the path specified in the create request is a **symbolic link**, the server MUST return an error as specified in section [2.2.2.1](#). Symbolic links MUST NOT be evaluated by the server.

If the final component of the path is a symbolic link, the server behavior depends on whether the flag FILE_OPEN_REPARSE_POINT was specified in the **CreateOptions** field of the request. If FILE_OPEN_REPARSE_POINT was specified, the server MUST open the underlying file or directory and return a handle to it. Otherwise, the server MUST return an error as specified in section [2.2.2.1](#).

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.

The server SHOULD [<184>](#) fail any request having a create context not specified in [2.2.13.2](#), with a STATUS_INVALID_PARAMETER error.

Open Execution:

If the `FILE_DELETE_ON_CLOSE` flag is set in **CreateOptions**, and the `DELETE` flag is not set in **DesiredAccess**, the server MUST fail the operation with `STATUS_INVALID_PARAMETER`.

The **ImpersonationLevel** in the request MUST have one of the values specified in section [2.2.13](#). Otherwise, the server MUST fail the request with `STATUS_BAD_IMPERSONATION_LEVEL`.

When opening a named pipe, if the **ImpersonationLevel** level is `Delegate`, the server MUST fail the request with `STATUS_BAD_IMPERSONATION_LEVEL`.

For open requests on a share of type `STYPE_DISKTREE` (as indicated by **TreeConnect.Share.Type**), the server MUST do the following:

- If **TreeConnect.Share.RestrictExclusiveOpens** is `TRUE` and the **ShareAccess** field does not include `FILE_SHARE_READ`, and the **DesiredAccess** field does not include `GENERIC_ALL`, `GENERIC_WRITE`, `FILE_WRITE_DATA`, `FILE_WRITE_ATTRIBUTES`, `FILE_WRITE_EA`, or `FILE_APPEND_DATA`, the server SHOULD [<185>](#) set `FILE_SHARE_READ` in the **ShareAccess** field.
- If **TreeConnect.Share.ForceSharedDelete** is `TRUE`, the server MUST set `FILE_SHARE_DELETE` in the **ShareAccess** field.
- If **TreeConnect.Share.ForceLevel2Oplock** is `TRUE`, and **RequestedOplockLevel** is `SMB2_OPLOCK_LEVEL_BATCH` or `SMB2_OPLOCK_LEVEL_EXCLUSIVE`, the server SHOULD [<186>](#) set **RequestedOplockLevel** to `SMB2_OPLOCK_LEVEL_II`.

The server MUST use the security context of the session in **Session.SecurityContext** to attempt to open the named object in the underlying object store using the parameters specified for **DesiredAccess**, **FileAttributes**, **ShareAccess**, **CreateDisposition**, **CreateOptions**, 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 [\[MS-FSA\].<187>](#) See section [2.2.13](#) for more details on the exact meaning of the various flags and options. If the underlying object store returns a failure for the attempted Open, the server MUST send an SMB2 error response with an error code as specified in section [2.2.2](#). The same rules apply when opening named pipe and print files, except that some flags and options are not supported when opening named pipes and print files. 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.<188>](#)

If the underlying object store returns `STATUS_ACCESS_DENIED`, **ServerStatistics.sts0_permerrors** MUST be increased by 1.

Successful Open Initialization:

If the open is successful, the server MUST allocate an open object for this open and insert it into **Session.OpenTable** and **GlobalOpenTable**. If **TreeConnect.Share.Type** is not equal to `STYPE_PRINTQ`, **ServerStatistics.sts0_fopens** MUST be increased by 1. If **TreeConnect.Share.Type** is equal to `STYPE_PRINTQ`, **ServerStatistics.sts0_jobsqueued** MUST be increased by 1. The server MUST also register the **Open** by invoking the event specified in [\[MS-](#)

[SRVS](#) section 3.1.6.4 and assign the return value to **Open.FileGlobalId**. The other 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 **MAXIMUM_ALLOWED** is included in the **DesiredAccess**.
- **Open.OplockLevel** is set to **SMB2_OPLOCK_LEVEL_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.
- **Open.CurrentEaIndex** is set to 1.
- **Open.CurrentQuotaIndex** is set to 1.
- **Open.TreeConnect** is set to refer to the **TreeConnect** on which the open request was performed and **Open.TreeConnect.OpenCount** MUST be increased by 1.
- **Open.LockCount** is set to 0.
- **Open.PathName** is set to the full local path that the current open is performed on.

If the server implements SMB 2.1 and supports leasing, the server MUST initialize the following:

- **Open.Lease** MUST be set to **NULL**.
- **Open.IsResilient** MUST be set to **FALSE**.
- **Open.ResilientOpenTimeout** MUST be set to 0.
- **Open.LockSequenceArray[]**: Each element of **Open.LockSequenceArray[]** MUST be set to 0.

The server MUST locate the **Request** in **Connection.RequestList** for which **Request.MessageId** matches the **MessageId** value in the SMB2 header, and set **Request.Open** to the **Open**.

Oplock Acquisition:

If the server does not implement SMB 2.1 or does not support leasing, and **RequestedOplockLevel** is set to SMB2_OPLOCK_LEVEL_LEASE, the request MUST be failed with STATUS_NOT_SUPPORTED.

If the server implements SMB 2.1 and supports leasing, and the **RequestedOplockLevel** is set to SMB2_OPLOCK_LEVEL_LEASE, the server MUST attempt to acquire a lease on the open from the underlying object store as described in section [3.3.5.9.8](#).

Otherwise, 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. [<189>](#) 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. [<190>](#)
- **LastAccessTime** is set to the value queried from the object store for when the object was last accessed. [<191>](#)
- **LastWriteTime** is set to the value queried from the object store for when the object was last written to. [<192>](#)
- **ChangeTime** is set to the value queried from the object store for when the object was last modified, including attribute changes. [<193>](#)
- **AllocationSize** is set to the amount of space reserved for the object, in bytes, on the underlying object store. [<194>](#) If this is a named pipe, **AllocationSize** SHOULD be 0. [<195>](#)
- **EndOfFile** is set to the size of the **main stream** of the object in bytes. [<196>](#) For named pipes this value SHOULD be 0. [<197>](#)
- **FileAttributes** MUST be set to the attributes of the object following the syntax specified in section [2.2.14](#). [<198>](#)
- **FileId.Persistent** MUST be set to **Open.DurableFileId**.
- **FileId.Volatile** MUST be set to **Open.FileId**.
- **CreateContextsLength** and **CreateContextsOffset** MUST be set to 0.

This response MUST be sent back to the client.

The status code returned by this operation MUST be one of those defined in [\[MS-ERREF\]](#). Common status codes returned by this operation include:

- STATUS_SUCCESS
- STATUS_INSUFFICIENT_RESOURCES
- STATUS_ACCESS_DENIED
- STATUS_OBJECT_NAME_NOT_FOUND
- STATUS_INVALID_PARAMETER
- STATUS_STOPPED_ON_SYMLINK
- STATUS_USER_SESSION_DELETED
- STATUS_NETWORK_NAME_DELETED
- STATUS_NETWORK_SESSION_EXPIRED
- STATUS_NOT_SUPPORTED
- STATUS_EAS_NOT_SUPPORTED
- STATUS_DISK_FULL
- STATUS_FILE_CLOSED

The following create contexts are potentially received as part of the create request. In each subsection, handling this create context is outlined. Any combination of the create contexts listed in the following sections is valid, with the following exception:

- When **SMB2_CREATE_DURABLE_HANDLE_RECONNECT**, [3.3.5.9.7](#), is received, the server SHOULD [<199>](#) ignore any other create contexts except SMB2_CREATE_REQUEST_LEASE in that request.

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. [<200>](#) 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.

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. [<201>](#) If the object store does not support

file security, the value MAY [<202>](#) be ignored or STATUS_NOT_SUPPORTED SHOULD 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. The server SHOULD support this create context request. [<203>](#) 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 requested space for the created file. [<204>](#) 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.

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. [<205>](#) 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. [<206>](#)

3.3.5.9.5 Handling the SMB2_CREATE_QUERY_MAXIMAL_ACCESS_REQUEST Create Context

The client is requesting that the server return maximal access information if the last modified time for the object that was opened, as returned by the underlying object store, is not equal to the time stamp provided by the client in the create context.

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. [<207>](#) It must construct an [SMB2_CREATE_QUERY_MAXIMAL_ACCESS_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.

If no time stamp is present in the request, the server MUST return maximal access information unconditionally.

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.

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](#). The server MUST also set **Open.DurableOwner** to a security descriptor accessible only by the user represented by **Open.Session.SecurityContext**.

In the "Response Construction" phase, the server MUST construct an [SMB2_CREATE_DURABLE_HANDLE_RESPONSE](#) response create context, following the syntax specified in section [2.2.14.2.3](#), and include it in the buffer described by the response **CreateContextLength** and **CreateContextOffset**.

3.3.5.9.7 Handling the SMB2_CREATE_DURABLE_HANDLE_RECONNECT Create Context

The client is requesting a reconnect to an existing durable or resilient open.

If an [SMB2_CREATE_REQUEST_LEASE create context](#) is also present in the request, and the server implements SMB 2.1 and supports leasing, and **Open.Lease** is not NULL, and **Lease.LeaseKey** does not match the **LeaseKey** provided in the SMB2_CREATE_REQUEST_LEASE create context, 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](#).

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 and **Open.IsResilient** is FALSE or unimplemented, 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 the user represented by **Session.SecurityContext** is not the same user denoted by **Open.DurableOwner**, the server MUST fail the request with STATUS_ACCESS_DENIED and proceed as specified in "Failed Open Handling" in section [3.3.5.9](#).
5. The server MUST set the **Open.Connection** to refer to the connection that received this request.
6. The server MUST set the **Open.Session** to refer to the session that received this request.
7. The server MUST set the **Open.TreeConnect** to refer to the tree connect that received this request, and **Open.TreeConnect.OpenCount** MUST be increased by 1.
8. The server MUST regenerate **Open.FileId** (the volatile portion of the [SMB2_FILEID](#)).

9. The server MUST insert the open into the **Session.OpenTable** with the **Open.FileId** as the new key.
10. The "Successful Open Initialization" and "Oplock Acquisition" phases MUST be skipped, and processing MUST continue as specified in "Response Construction".
11. In the "Response Construction" phase, if the server implements SMB 2.1 and supports leasing, and **Connection.Dialect** is equal to "2.100", and **Open.Lease** is not NULL, then the server MUST construct an [SMB2_CREATE_RESPONSE_LEASE](#) create context, following the syntax specified in section [2.2.14.2.9](#), and include it in the buffer described by the response **CreateContextLength** and **CreateContextOffset**. This structure MUST have the following values set:
 - **LeaseKey** MUST be set to **Lease.LeaseKey**.
 - **LeaseState** MUST be set to **Lease.LeaseState**.

3.3.5.9.8 Handling the SMB2_CREATE_REQUEST_LEASE Create Context

This section applies only to servers that implement SMB 2.1.

If both SMB2_CREATE_DURABLE_HANDLE_RECONNECT and SMB2_CREATE_REQUEST_LEASE create contexts are present in the request, they are processed as specified in section [3.3.5.9.7](#), and this section does not apply.

If the server does not implement SMB 2.1 and support leasing, or **Connection.Dialect** is not equal to "2.100" or if the **RequestedOplockLevel** is not SMB2_OPLOCK_LEVEL_LEASE, the server MUST ignore the [SMB2_CREATE_REQUEST_LEASE Create Context](#) request.

By specifying a **RequestedOplockLevel** of SMB2_OPLOCK_LEVEL_LEASE, the client is requesting that a lease be acquired for this open. If the request does not provide an SMB2_CREATE_REQUEST_LEASE Create Context, the lease request MUST be ignored and **Open.OplockLevel** MUST be set to SMB2_OPLOCK_LEVEL_NONE.

The processing changes involved in acquiring the lease are:

In the "Path Name Validation" phase, the server MUST attempt to locate a Lease Table by performing a lookup in **GlobalLeaseTableList** using **Connection.ClientGuid** as the lookup key. If no **LeaseTable** is found, one MUST be allocated and the following values set:

- **LeaseTable.ClientGuid** is set to **Connection.ClientGuid**.
- **LeaseTable.LeaseList** is set to an empty list.

If the allocation fails, the create request MUST be failed with STATUS_INSUFFICIENT_RESOURCES.

The server MUST attempt to locate a Lease by performing a lookup in the **LeaseTable.LeaseList** using the **LeaseKey** in the [SMB2_CREATE_REQUEST_LEASE](#) as the lookup key. If a lease is found but **Lease.Filename** does not match the file name for the incoming request, the request MUST be failed with STATUS_INVALID_PARAMETER.

If no lease is found, one MUST be allocated with the following values set:

- **Lease.LeaseKey** is set to the **LeaseKey** in the **SMB2_CREATE_REQUEST_LEASE** create context.
- **Lease.Filename** is set to the file being opened.

- **Lease.LeaseState** is set to NONE.
- **Lease.BreakToLeaseState** is set to NONE.
- **Lease.LeaseBreakTimeout** is set to 0.
- **Lease.LeaseOpens** is set to an empty list.
- **Lease.Breaking** is set to FALSE.

If the allocation fails, the create request MUST be failed with STATUS_INSUFFICIENT_RESOURCES. Otherwise, if a **LeaseTable** was created it MUST be added to the **GlobalLeaseTableList**, and if a Lease was created it MUST be added to the **LeaseTable.LeaseList**.

At this point, execution of create continues as described in [3.3.5.9](#) until the Oplock Acquisition phase.

The caching state requested in **LeaseState** of the SMB2_CREATE_REQUEST_LEASE SHOULD contain a valid **LeaseState** as specified in [3.3.1.12](#). The server MUST ignore the undefined bits in **LeaseState**.

During "Oplock Acquisition", if the underlying object store does not support leasing, the server SHOULD fall back to requesting a batch oplock instead of a lease and continue processing as described in "Oplock Acquisition". If the underlying object store does support leasing, the following steps are taken:

If **TreeConnect.Share.ForceLevel2Oplock** is TRUE, and **LeaseState** includes SMB2_LEASE_WRITE_CACHING, the server MUST clear the bit SMB2_LEASE_WRITE_CACHING in the **LeaseState** field.

If the caching state requested in **LeaseState** of the **SMB2_CREATE_REQUEST_LEASE** is not a superset of **Lease.LeaseState** or if **Lease.Breaking** is TRUE, the server MUST NOT promote **Lease.LeaseState**. If the lease state requested is a superset of **Lease.LeaseState** and **Lease.Breaking** is FALSE, the server MUST request promotion of the lease state from the underlying object store to the new caching state. [<208>](#) If the object store succeeds this request, **Lease.LeaseState** MUST be set to the new caching state. If **Lease.Breaking** is TRUE, the server MUST return the existing **Lease.LeaseState** to client and set **LeaseFlags** to be SMB2_LEASE_FLAG_BREAK_IN_PROGRESS. At this point, execution continues as described in section [3.3.5.9](#) until the "Response Construction" phase.

In the "Response Construction" phase, the server MUST construct an [SMB2_CREATE_RESPONSE_LEASE](#) response create context, following the syntax specified in section [2.2.14.2.9](#), and include it in the buffer described by the response **CreateContextLength** and **CreateContextOffset**. This structure MUST have the following values set:

- **LeaseKey** MUST be set to **Lease.LeaseKey**.
- **LeaseState** MUST be set to **Lease.LeaseState**.

The server MUST set **Open.OplockState** to Lease, set **Open.Lease** to a reference to Lease, set **Open.OplockLevel** to SMB2_OPLOCK_LEVEL_LEASE, and add Open to **Lease.LeaseOpens**. The remainder of open response construction continues as described in "Response Construction".

3.3.5.9.9 Handling the SMB2_CREATE_QUERY_ON_DISK_ID Create Context

The server generates a 32-byte value that the client can use to identify the open file. [<209>](#209) The value MUST be chosen such that the same value is not assigned to any other file on the server as long as the client holds the file open.

The processing changes involved for this create context are as follows:

In the "Response Construction" phase, the server MUST set **DiskIDBuffer** to the generated 32-byte value. It MUST construct an SMB2_CREATE_QUERY_ON_DISK_ID create context, following the syntax specified in section [2.2.14.2.8](#2.2.14.2.8), and include it in the buffer described by the response's **CreateContextLength** and **CreateContextOffset**.

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 **GlobalSessionTable**. 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.

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. Otherwise, the server MUST locate the **Request** in **Connection.RequestList** for which **Request.MessageId** matches the **MessageId** value in the SMB2 header, set **Request.Open** to the **Open**, and close the Open as specified in section [3.3.4.17](#3.3.4.17).

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. [<210>](#210) 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](#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.

The status code returned by this operation MUST be one of those defined in [\[MS-ERREF\]](#). Common status codes returned by this operation include:

- STATUS_SUCCESS
- STATUS_INSUFFICIENT_RESOURCES
- STATUS_FILE_CLOSED
- STATUS_NETWORK_NAME_DELETED
- STATUS_USER_SESSION_DELETED
- STATUS_INVALID_PARAMETER
- STATUS_NETWORK_SESSION_EXPIRED
- STATUS_ACCESS_DENIED

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 **GlobalSessionTable**. 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.

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. Otherwise, the server MUST locate the **Request** in **Connection.RequestList** for which **Request.MessageId** matches the **MessageId** value in the SMB2 header, and set **Request.Open** to the **Open**.

The server MUST issue a request to the underlying object store to flush any cached data for **Open.LocalOpen**.[<211>](#) 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[<212>](#) choose to handle this request asynchronously, as specified in section [3.3.4.2](#).)

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.

The status code returned by this operation MUST be one of those defined in [\[MS-ERREF\]](#). Common status codes returned by this operation include:

- STATUS_SUCCESS
- STATUS_INSUFFICIENT_RESOURCES
- STATUS_ACCESS_DENIED
- STATUS_FILE_CLOSED
- STATUS_NETWORK_NAME_DELETED
- STATUS_USER_SESSION_DELETED
- STATUS_NETWORK_SESSION_EXPIRED
- STATUS_INVALID_PARAMETER
- STATUS_PIPE_BROKEN
- STATUS_DISK_FULL
- STATUS_CANCELLED

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 **GlobalSessionTable**. 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.

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. Otherwise, the server MUST locate the **Request** in **Connection.RequestList** for which **Request.MessageId** matches the **MessageId** value in the SMB2 header, and set **Request.Open** to the **Open**.

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 SHOULD [<213>](#) fail the request with STATUS_INVALID_PARAMETER.

If **Connection.SupportsMultiCredit** is TRUE the server MUST validate **CreditCharge** based on **Length**, as specified in section [3.3.5.2.4](#). If the validation fails, it MUST fail the read 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 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**.<214>

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<215> be required to handle it asynchronously, as specified in section 3.3.4.2. To query a pipe's blocking mode, use the FilePipeInformation file information class, as specified in [MS-FSCC] section 2.4.29. To change a pipe's blocking mode, use an SMB2 SET_INFO Request with the FilePipeInformation file information class, as specified in [MS-FSCC] section 2.4.29.<216> If the read is not finished in 0.5 milliseconds, the server MUST send an interim response to the client.

If the read fails, the server MUST fail the request using the error code received from the read operation. If the read returns fewer bytes than specified by the **MinimumCount** field of the request, the server MUST fail the request with STATUS_END_OF_FILE.

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 zero.
- **DataOffset** MUST be set to the offset into the response in bytes from the beginning of the SMB2 header where the data is located.
- The data MUST be copied into the response.

The response MUST then be sent to the client.

The status code returned by this operation MUST be one of those defined in [MS-ERREF]. Common status codes returned by this operation include:

- STATUS_SUCCESS
- STATUS_INSUFFICIENT_RESOURCES
- STATUS_ACCESS_DENIED
- STATUS_FILE_CLOSED
- STATUS_NETWORK_NAME_DELETED
- STATUS_USER_SESSION_DELETED
- STATUS_NETWORK_SESSION_EXPIRED
- STATUS_INVALID_PARAMETER
- STATUS_END_OF_FILE
- STATUS_PIPE_BROKEN
- STATUS_BUFFER_OVERFLOW

- STATUS_CANCELLED
- STATUS_FILE_LOCK_CONFLICT

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 **GlobalSessionTable**. 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.

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. Otherwise, the server MUST locate the **Request** in **Connection.RequestList** for which **Request.MessageId** matches the **MessageId** value in the SMB2 header, and set **Request.Open** to the **Open**.

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 SHOULD [<217>](#) 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 SHOULD [<218>](#) fail the request with STATUS_INVALID_PARAMETER.

If **Connection.SupportsMultiCredit** is TRUE the server MUST validate **CreditCharge** based on **Length**, as specified in section [3.3.5.2.4](#). If the validation fails, it MUST fail the write 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**. If SMB 2.1 is supported, and SMB2_WRITEFLAG_WRITE_THROUGH is set in the **Flags** field of the SMB2 WRITE Request, the server SHOULD [<219>](#) indicate to the underlying object store that the write is to be written to persistent storage before completion is returned.

If the write 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 [<220>](#) be required to handle it asynchronously, as specified in section [3.3.4.2](#). To query a pipe's blocking mode, use the FilePipeInformation file information class, as specified in [\[MS-FSCC\]](#) section 2.4.29. To change a pipe's blocking mode, use an SMB2 [SMB2 SET_INFO Request](#) with the FilePipeInformation file information class, as specified in [\[MS-FSCC\]](#) section 2.4.29.

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 number of bytes written.
- **Remaining** MUST be set to zero.
- **WriteChannelInfoOffset** MUST be set to zero.
- **WriteChannelInfoLength** MUST be set to zero.

The response MUST then be sent to the client.

The status code returned by this operation MUST be one of those defined in [\[MS-ERREF\]](#). Common status codes returned by this operation include:

- STATUS_SUCCESS
- STATUS_INSUFFICIENT_RESOURCES
- STATUS_ACCESS_DENIED
- STATUS_FILE_CLOSED
- STATUS_NETWORK_NAME_DELETED
- STATUS_USER_SESSION_DELETED
- STATUS_NETWORK_SESSION_EXPIRED
- STATUS_INVALID_PARAMETER
- STATUS_PIPE_BROKEN
- STATUS_DISK_FULL
- STATUS_CANCELLED
- STATUS_FILE_LOCK_CONFLICT

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 **GlobalSessionTable**. 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.

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. Otherwise, the server MUST locate the **Request** in **Connection.RequestList** for which **Request.MessageId** matches the **MessageId** value in the SMB2 header, and set **Request.Open** to the **Open**.

If the negotiate dialect is SMB 2.1, the server supports leasing, and **Open.IsResilient** is TRUE, the server MUST verify whether the lock/unlock request with the same **LockSequence** has been successfully processed before through the following steps:

1. The server MUST compute $((\text{LockSequence} \gg 4) - 1)$ to use as an index into **Open.LockSequenceArray[]** in order to locate the sequence number entry (1 byte). If the index exceeds the maximum extent of the **Open.LockSequenceArray[]**, the server MUST skip step 2 and continue lock/unlock processing.
2. The server MUST take the 4 least significant bits of **LockSequence** as the lock sequence number and compare to the entry retrieved from step 1. If the sequence numbers are equal, the server MUST complete the lock/unlock request with success. Otherwise, the server MUST reset the entry value to be 0xFF and continue lock/unlock processing.

If the flags of the initial [SMB2_LOCK_ELEMENT](#) 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.

The status code returned by this operation MUST be one of those defined in [\[MS-ERREF\]](#). Common status codes returned by this operation include:

- STATUS_SUCCESS
- STATUS_INSUFFICIENT_RESOURCES
- STATUS_ACCESS_DENIED
- STATUS_FILE_CLOSED
- STATUS_NETWORK_NAME_DELETED
- STATUS_USER_SESSION_DELETED
- STATUS_NETWORK_SESSION_EXPIRED
- STATUS_INVALID_PARAMETER
- STATUS_FILE_LOCK_CONFLICT
- STATUS_CANCELLED
- STATUS_LOCK_NOT_GRANTED

3.3.5.14.1 Processing Unlocks

For each [SMB2_LOCK_ELEMENT](#) 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, and all successfully processed unlock operations will not be rolled back.

If SMB2_LOCKFLAG_FAIL_IMMEDIATELY is set, the server MAY [<221>](#) ignore this flag.

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_ELEMENT entry. [<222>](#) 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, the server MUST decrease **Open.LockCount** by 1. If 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, and the negotiate dialect is SMB 2.1, the server supports leasing, and **Open.IsResilient** is TRUE, the server MUST set the lock sequence number in **Open.LockSequenceArray[]** through the following steps to indicate the unlock request with **LockSequence** has been successfully processed by the server:

1. The server MUST compute $((\text{LockSequence} \gg 4) - 1)$ to use as an index into **Open.LockSequenceArray[]** in order to locate the sequence number entry (1 byte). If the index exceeds the maximum extent of the **Open.LockSequenceArray[]**, the server MUST skip step 2 and continue unlock processing.
2. The server MUST take the 4 least significant bits of **LockSequence** as the lock sequence number and save it into the corresponding entry located from step 1.

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

If the **Locks** array has more than one entry and the **Flags** field in any of these entries does not have SMB2_LOCKFLAG_FAIL_IMMEDIATELY set, the server SHOULD [<223>](#) fail the request with STATUS_INVALID_PARAMETER. 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.

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_ELEMENT entry. [<224>](#) 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. [<225>](#)

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 **Locks** array of this request. It MUST decrement **Open.LockCount** by the number of locks unlocked. It MUST stop processing any remaining entries in the **Locks** array and MUST fail the operation with the error code received from the lock operation.

If the lock operation succeeds, the server MUST increase **Open.LockCount** by 1. If there are remaining entries in the **Locks** array, the server MUST continue processing the next entry in the **Locks** array as described previously.

If the lock operation succeeds, and there are no remaining entries in the **Locks** array, and the negotiate dialect is SMB 2.1, the server supports leasing, and **Open.IsResilient** is TRUE, the server MUST set the lock sequence number in **Open.LockSequenceArray[]** through the following steps to indicate the lock request with **LockSequence** has been successfully processed by the server:

1. The server MUST compute $((\text{LockSequence} \gg 4) - 1)$ to use as an index into **Open.LockSequenceArray[]** in order to locate the sequence number entry (1 byte). If the index exceeds the maximum extent of the **Open.LockSequenceArray[]**, the server MUST skip step 2 and continue lock processing.
2. The server MUST take the 4 least significant bits of **LockSequence** as the lock sequence number and save it into the corresponding entry located from step 1.

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 **GlobalSessionTable**. 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.

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 **Flags** field of the request is not SMB2_0_IOCTL_IS_FSCTL the server MUST fail the request with STATUS_NOT_SUPPORTED.

If the **CtlCode** is 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. Otherwise, the server MUST locate the **Request** in **Connection.RequestList** for which **Request.MessageId** matches the **MessageId** value in the SMB2 header, and set **Request.Open** to the **Open**.

The server MUST fail the request with STATUS_INVALID_PARAMETER in the following cases:

- If **InputOffset** is greater than zero but less than (size of SMB2 Header + size of the SMB2 IOCTL request not including **Buffer**) or if **InputOffset** is greater than (size of SMB2 Header + size of the SMB2 IOCTL request).
- If **OutputOffset** is greater than zero but less than (size of SMB2 Header + size of the SMB2 IOCTL request not including **Buffer**) or if **OutputOffset** is greater than (size of SMB2 Header + size of the SMB2 IOCTL request).

- If (**InputOffset** + **InputCount**) is greater than (size of SMB2 Header + size of the SMB2 IOCTL request).
- If (**OutputOffset** + **OutputCount**) is greater than (size of SMB2 Header + size of the SMB2 IOCTL request).
- If **OutputCount** is greater than zero and **OutputOffset** is less than (**InputOffset** + **InputCount**).

Note that any padding inserted in the response 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.

The server MUST NOT return an output buffer containing more bytes of data than the **MaxOutputResponse** value specified by the client. If the underlying object store indicates an insufficient buffer passed in with STATUS_BUFFER_OVERFLOW, the server SHOULD set the **OutputCount** in the IOCTL response structure to the size of the data returned in that buffer by the underlying object store and SHOULD [<226>](#) copy **OutputCount** bytes into the output buffer, and MUST return a status of STATUS_BUFFER_OVERFLOW.

If **Connection.SupportsMultiCredit** is TRUE, the server MUST validate **CreditCharge** based on the maximum of (**InputCount** + **OutputCount**) and (**MaxInputResponse** + **MaxOutputResponse**), as specified in section [3.3.5.2.4](#). If the validation fails, it MUST fail the IOCTL request with STATUS_INVALID_PARAMETER.

A server SHOULD [<227>](#) return a status of STATUS_INVALID_DEVICE_REQUEST when an FSCTL is not supported remotely or is not supported on the file system on which the file or directory handle specified by the FSCTL exists, as specified in [\[MS-FSCC\]](#) section 2.2.

Processing for a specific **CtlCode** is as specified in subsequent sections.

The status code returned by this operation MUST be one of those defined in [\[MS-ERREF\]](#). Common status codes returned by this operation include:

- STATUS_SUCCESS
- STATUS_INSUFFICIENT_RESOURCES
- STATUS_ACCESS_DENIED
- STATUS_FILE_CLOSED
- STATUS_NETWORK_NAME_DELETED
- STATUS_USER_SESSION_DELETED
- STATUS_NETWORK_SESSION_EXPIRED
- STATUS_CANCELLED
- STATUS_INVALID_PARAMETER
- STATUS_BUFFER_OVERFLOW
- STATUS_NOT_SUPPORTED
- STATUS_BUFFER_TOO_SMALL

- STATUS_OBJECT_NAME_NOT_FOUND
- STATUS_END_OF_FILE
- STATUS_INVALID_DEVICE_REQUEST

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. [<228>](#) 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](#), and SHOULD [<229>](#) set **NumberOfSnapShots** to zero, **NumberOfSnapShotsReturned** to zero, and **SnapShotArraySize** to zero.

If there are previous versions available, the server MUST calculate the size required to return the SRV_SNAPSHOT_ARRAY structure containing the previous version array. If the size 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 **SnapShots[]** array size required to receive all of the previous version timestamps. If the maximum output buffer size specified is too small to contain this SRV_SNAPSHOT_ARRAY, the server MUST return the status STATUS_INVALID_PARAMETER.

If the size of the SRV_SNAPSHOT_ARRAY structure containing the previous version array 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 the offset, in bytes, from the beginning of the SMB2 header to the **Buffer[]** field of the response.
- **InputCount** SHOULD be set to zero.
- **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 zero.

- 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:

If **IsDfsCapable** is set to FALSE, the server MUST return STATUS_FS_DRIVER_REQUIRED to the client.

The server MUST pass the following to Distributed File System (DFS):

- The value of the **MaxReferralLevel** field from the REQ_GET_DFS_REFERRAL structure, as specified in [\[MS-DFSC\]](#) section 2.2.2, that was received in this request.
- The target name in the REQ_GET_DFS_REFERRAL structure.
- The IP address of the client.
- The maximum size of the response data buffer that will be accepted by the client.

The processing of this request by the DFS server is as specified in [\[MS-DFSC\]](#) sections [3.2.5.1](#) and [3.3.5.1](#).

If DFS returns a failure, the server MUST fail the request with the error code received from DFS. If the error returned from DFS is STATUS_BUFFER_OVERFLOW, the server SHOULD [<230>](#) copy the data returned by DFS into a normal FSCTL_GET_DFS_REFERRALS response and return STATUS_BUFFER_OVERFLOW to the client as noted in sections [3.3.4.4](#) and [3.3.5.15](#).

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 the offset, in bytes, from the beginning of the SMB2 header to the **Buffer[]** field of the response.
- **InputCount** SHOULD be set to zero.
- **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 zero.
- 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.

If the share on which the request is being executed is not a named pipe share, the server SHOULD [<231>](#) fail the request with STATUS_NOT_SUPPORTED.

The server MUST attempt to write the number of bytes specified in the request by the **InputCount** field 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. For more information on reading from a pipe, see section [3.3.5.12](#).

If the read/write attempt is not finished in 1 millisecond, the server MUST send an interim response to the client. If the read/write attempt succeeds, [<232>](#) 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 the offset, in bytes, from the beginning of the SMB2 header to the **Buffer[]** field of the response.
- **InputCount** SHOULD [<233>](#) be set to zero.
- If any data was read from the pipe, **OutputOffset** MUST be set to **InputOffset** + **InputCount**, rounded up to a multiple of 8. Otherwise, **OutputOffset** SHOULD [<234>](#) be set to zero.
- **OutputCount** MUST be set to the number of bytes read from the pipe. If no data is to be returned, the server MUST set **OutputCount** to zero.
- **Flags** MUST be set to zero.
- The server MUST copy the bytes read into the **Buffer** field at the **OutputOffset** computed above.

The response MUST be sent to the client.

3.3.5.15.4 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. A **MaxOutputResponse** value of zero is allowed.

If the share on which the request is being executed is not a named pipe share, the server SHOULD [<235>](#) fail the request with STATUS_NOT_SUPPORTED.

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 the offset, in bytes, from the beginning of the SMB2 header to the **Buffer[]** field of the response.
- **InputCount** SHOULD be set to zero.
- If any data was read from the pipe, **OutputOffset** MUST be set to **InputOffset** + **InputCount**, rounded up to a multiple of 8. Otherwise, **OutputOffset** SHOULD [<236>](#) be set to zero.
- **OutputCount** MUST be set to the number of bytes read from the pipe.
- **Flags** MUST be set to zero.
- 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.<237>](#)

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. [<238>](#)

If the maximum output buffer size specified is too small to contain an **SRV_REQUEST_RESUME_KEY** structure, the server MUST return the status `STATUS_INVALID_PARAMETER`.

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 the offset, in bytes, from the beginning of the SMB2 header to the **Buffer[]** field of the response.
- **InputCount** SHOULD be set to zero.
- **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 zero.
- 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 or FSCTL_SRV_COPYCHUNK_WRITE, 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.GrantedAccess** of the destination file does not include FILE_WRITE_DATA, then the request MUST be failed with STATUS_ACCESS_DENIED. If **Open.GrantedAccess** of the destination file does not include FILE_READ_DATA access and the **CtlCode** is FSCTL_SRV_COPYCHUNK, then the request MUST be failed with STATUS_ACCESS_DENIED.

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 [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. [<239>](#) If the **SourceOffset** or **SourceOffset + Length** extends beyond the end of file, the server SHOULD [<240>](#) 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 send an SMB2 IOCTL response as specified in [Sending a Copy Failure Server-Side Copy Response \(section 3.3.5.15.6.1\)](#).

The server MUST issue a write using the **TargetOffset** and **Length** in the range against the destination file. [<241>](#) If the write fails, the server MUST send an SMB2 IOCTL response as specified in [Sending a Copy Failure Server-Side Copy Response \(section 3.3.5.15.6.1\)](#).

If all ranges are copied successfully, the server MUST construct an SMB2 IOCTL Response following the syntax specified in the section [2.2.32](#), with the following values:

- **CtlCode** MUST be set to FSCTL_SRV_COPYCHUNK or FSCTL_SRV_COPYCHUNK_WRITE.
- **FileId.Persistent** MUST be set to **Open.DurableFileId**. **FileId.Volatile** MUST be set to **Open.FileId**.
- **InputOffset** SHOULD be set to the offset, in bytes, from the beginning of the SMB2 header to the **Buffer[]** field of the response.
- **InputCount** SHOULD be set to zero.
- **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 zero.
- 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. **ChunksWritten** MUST be set to the number of chunks processed. **ChunkBytesWritten** MUST be set to zero. **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 Sending a Copy Failure Server-Side 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 [2.2.32](#), with the following values:

- **Status** in the [SMB2 header](#) MUST be set to one of the error codes listed in section [3.3.5.15](#).
- **CtlCode** MUST be set to FSCTL_SRV_COPYCHUNK or FSCTL_SRV_COPYCHUNK_WRITE.
- **FileId.Persistent** MUST be set to **Open.DurableFileId**. **FileId.Volatile** MUST be set to **Open.FileId**.
- **InputOffset** SHOULD be set to the offset, in bytes, from the beginning of the SMB2 header to the **Buffer[]** field of the response.
- **InputCount** SHOULD be set to zero.
- **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 zero.
- 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. **ChunksWritten** MUST be set to the number of chunks successfully written. If the error was encountered partway through a write, **ChunkBytesWritten** MUST be set to the number of bytes written in the final, partial write. Otherwise, **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.

3.3.5.15.6.2 Sending an Invalid Parameter Server-Side Copy Response

If the server determines that the total chunk count is more than **ServerSideCopyMaxNumberOfChunks**, or the size of any chunk is more than **ServerSideCopyMaxChunkSize**, or the total size of all chunks exceeds **ServerSideCopyMaxDataSize**, 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 or FSCTL_SRV_COPYCHUNK_WRITE.

- **FileId.Persistent** MUST be set to **Open.DurableFileId**. **FileId.Volatile** MUST be set to **Open.FileId**.
- **InputOffset** SHOULD be set to the offset, in bytes, from the beginning of the SMB2 header to the **Buffer[]** field of the response.
- **InputCount** SHOULD be set to zero.
- **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 zero.
- 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 **ServerSideCopyMaxNumberOfChunks**. **ChunkBytesWritten** MUST be set **ServerSideCopyMaxChunkSize**. **TotalBytesWritten** MUST be set to **ServerSideCopyMaxDataSize**.

The response MUST be sent to the client.

3.3.5.15.7 Handling a Content Information Retrieval 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_READ_HASH, message handling proceeds as follows:

The server MUST fail the [SRV_READ_HASH request](#) with the error code specified in the following cases:

- If the server does not support SRV_READ_HASH requests, it MUST fail the request with STATUS_NOT_SUPPORTED. [<242>](#)
- If the server supports SRV_READ_HASH requests but does not have the branch cache feature available, it MUST fail the request with STATUS_HASH_NOT_PRESENT.
- If **InputCount** in the request is less than the size of a SRV_READ_HASH request or if **MaxOutputResponse** in the request is less than the size of a [SRV_READ_HASH Response](#), the server MUST fail the request with error STATUS_BUFFER_TOO_SMALL.
- If the **HashType** field of the SRV_READ_HASH request is not equal to SRV_HASH_TYPE_PEER_DIST, or the **HashVersion** is not equal to SRV_HASH_VER_1, or the **HashRetrievalType** is not equal to SRV_HASH_RETRIEVE_HASH_BASED, the server MUST fail the SRV_READ_HASH request with error STATUS_INVALID_PARAMETER.
- If **ServerHashLevel** is **HashDisableAll**, the server MUST fail the SRV_READ_HASH request with error code STATUS_HASH_NOT_SUPPORTED.
- If **ServerHashLevel** is **HashEnableShare** and **Open.TreeConnect.Share.HashEnabled** is FALSE, the server MUST fail the SRV_READ_HASH request with error code STATUS_HASH_NOT_SUPPORTED.

The server MUST open the Content Information File from the object store for the object represented by **Open.LocalOpen** with specified offset. If the Content Information File open fails, the server MUST fail the SRV_READ_HASH request with the error code returned by object store.

If the Content Information File open succeeds, the server MUST verify the following:

- If the Content Information File is empty, the server MUST fail the SRV_READ_HASH request with the error code STATUS_HASH_NOT_PRESENT.
- If the **Offset** field of the SRV_READ_HASH request is equal to or beyond the end of the Content Information File, the server MUST fail the SRV_READ_HASH request with error code STATUS_END_OF_FILE.
- The Content Information File MUST start with a valid [HASH_HEADER](#) as specified in section [2.2.31.2.1](#).
 - If the **HashType** field in the HASH_HEADER is not equal to the **HashRetrievalType** field of the SRV_READ_HASH request, the server MUST fail the SRV_READ_HASH request with the error code STATUS_HASH_NOT_PRESENT.
 - If the **HashVersion** field in the HASH_HEADER is not equal to the **HashVersion** field of the SRV_READ_HASH request, the server MUST fail the SRV_READ_HASH request with the error code STATUS_HASH_NOT_PRESENT.
 - If the **Dirty** field in the HASH_HEADER is a nonzero value, the server MUST fail the SRV_READ_HASH request with the error code STATUS_HASH_NOT_PRESENT.

If the Content Information File is verified successfully, 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_READ_HASH.
- **FileId.Persistent** MUST be set to **Open.DurableFileId**.
- **FileId.Volatile** MUST be set to **Open.FileId**.
- **InputOffset** SHOULD be set to the offset, in bytes, from the beginning of the SMB2 header to the **Buffer[]** field of the response.
- **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 size of SRV_READ_HASH Response, including the variable length for Content Information.
- **Flags** MUST be set to zero.
- The server MUST copy a SRV_READ_HASH Response following the syntax specified in section [2.2.32.4](#) into the **Buffer** field at the **OutputOffset** computed above.

3.3.5.15.8 Handling a Pass-Through Operation Request

Pass-through requests are I/O Control requests and File System Control (FSCTL) requests with a **CtlCode** value that is not specified in section [2.2.31](#). As noted in section [3.3.5.15](#), the server MUST fail I/O Control requests with STATUS_NOT_SUPPORTED.

Pass-through FSCTL requests fall further into two types, those for which a **CtlCode** value matches an FSCTL function number defined in [\[MS-FSCC\]](#) section 2.3, and those that do not. When the latter type of pass-through request does not meet the private FSCTL requirements of [\[MS-FSCC\]](#) section 2.3, the server MUST NOT pass the request to the underlying object store and MUST fail the request by sending a response of STATUS_NOT_SUPPORTED.

Otherwise, when the server receives a pass-through FSCTL request, the server SHOULD [<243>](#) pass it through to the underlying object store.

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. Where the **CtlCode** value matches an FSCTL function number defined in [MS-FSCC], the server SHOULD verify that the above buffers and sizes conform to the requirements of the corresponding structures defined in [\[MS-FSCC\]](#) section 2.3, and use the **FileId** from the SMB2 IOCTL request to obtain the handle described in [\[MS-FSCC\]](#) section 2.3 to pass to the object store. Where the **CtlCode** value is not defined in [MS-FSCC], the server SHOULD [<244>](#) ensure that the other requirements for private FSCTLs defined in [MS-FSCC] are met.

If the underlying object store returns a failure, the server MUST fail the request and send a response with an error code, as specified in [\[MS-ERREF\]](#) section 2.2.

Note that a successful FSCTL pass-through request could return 0 bytes of output buffer data, and have **OutputCount** set to 0. Similarly, it is possible for a valid FSCTL pass-through request to send 0 bytes of input buffer data, depending on the requirements of the FSCTL.

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**.
- **InputOffset** SHOULD be set to the offset, in bytes, from the beginning of the [SMB2 header](#) to the **Buffer[]** field of the response.
- **InputCount** MUST be set to the number of input bytes the object store is returning to the client.
- If the object store is returning output data to the client, **OutputOffset** MUST be set to **InputOffset** + **InputCount**, rounded up to a multiple of 8. Otherwise, **OutputOffset** SHOULD [<245>](#) be set to zero.
- 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 zero.
- 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.

3.3.5.15.9 Handling a Resiliency Request

When the server receives a request with an SMB2 header with a **Command** value equal to SMB2 IOCTL and a **CtlCode** [FSCTL_LMR_REQUEST_RESILIENCY](#), message handling proceeds as follows.

If the server does not support FSCTL_LMR_REQUEST_RESILIENCY requests, it MUST fail the request with STATUS_NOT_SUPPORTED. [<246>](#)

If **InputCount** is smaller than the size of the **NETWORK_RESILIENCY_REQUEST** request as specified in section [2.2.31.3](#), or if the requested **Timeout** is greater than **MaxResiliencyTimeout**, the request MUST be failed with STATUS_INVALID_PARAMETER.

Open.IsDurable MUST be set to FALSE. **Open.IsResilient** MUST be set to TRUE.

Open.ResiliencyTimeout MUST be set to the value of **Timeout** specified in NETWORK_RESILIENCY_REQUEST of the request. **Open.DurableOwner** MUST be set to a security descriptor accessible only by the user represented by **Open.Session.SecurityContext**.

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_LMR_REQUEST_RESILIENCY.
- **FileId.Persistent** MUST be set to **Open.DurableFileId**. **FileId.Volatile** MUST be set to **Open.FileId**.
- **InputOffset** SHOULD be set to the offset, in bytes, from the beginning of the SMB2 header to the **Buffer[]** field of the response.
- **InputCount** SHOULD be set to zero.
- **OutputOffset** MUST be set to **InputOffset + InputCount**, rounded up to a multiple of 8.
- **OutputCount** MUST be set to zero.
- **Flags** MUST be set to zero.

The response MUST be sent to the client.

3.3.5.15.10 Handling a Pipe Wait 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_WAIT, message handling proceeds as follows.

The server MUST ensure that the **Name** field of the FSCTL_PIPE_WAIT request identifies a named pipe. If the **Name** field is malformed, or no such object exists, the server MUST fail the request with STATUS_OBJECT_NAME_NOT_FOUND. If an object of that name exists, but it is not a named pipe, the server MUST fail the request with **STATUS_INVALID_DEVICE_REQUEST**.

The server MUST attempt to wait for a connection to the specified named pipe. If **TimeoutSpecified** is TRUE in the FSCTL_PIPE_WAIT request, the server MUST wait for the amount of time specified in the **Timeout** field in the FSCTL_PIPE_WAIT request for a connection to the named pipe. If no connection is available within the specified time, the server MUST fail the request with STATUS_IO_TIMEOUT. If **TimeoutSpecified** is FALSE, the server MUST wait forever for a connection to the named pipe.

If a connection to the specified named pipe is available, the server MUST construct an [SMB2 IOCTL Response](#) by following the syntax specified in section [2.2.32](#), with the exception of the following values:

- The **CtlCode** field MUST be set to FSCTL_PIPE_WAIT.
- The **FileId** field MUST be set to { 0xFFFFFFFFFFFFFFFF, 0xFFFFFFFFFFFFFFFF }.
- The **OutputCount** field MUST be set to 0.

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](#).

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** where **Request.AsyncId** matches the **AsyncId** of the incoming cancel request. If SMB2_FLAGS_ASYNC_COMMAND is not set, then the server MUST search for a request in **Connection.RequestList** where **Request.MessageId** 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. [<247>](#) 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 is required 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.

The status code returned by this operation MUST be one of those defined in [\[MS-ERREF\]](#). Common status codes returned by this operation include:

- STATUS_SUCCESS
- STATUS_INVALID_PARAMETER

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 **GlobalSessionTable**. 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.

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. Otherwise, the server MUST locate the **Request** in **Connection.RequestList** for which **Request.MessageId** matches the **MessageId** value in the SMB2 header, and set **Request.Open** to the **Open**.

If the open is not an open to a directory, the request MUST be failed with STATUS_NOT_SUPPORTED.

If **OutputBufferLength** is greater than **Connection.MaxTransactSize**, the server MUST fail the request with STATUS_INVALID_PARAMETER.

If **Connection.SupportsMultiCredit** is TRUE, the server MUST validate **CreditCharge** based on **OutputBufferLength**, as specified in section [3.3.5.2.4](#). If the validation fails, it MUST fail the request with STATUS_INVALID_PARAMETER.

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 classes 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 SHOULD [<248>](#) 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**. If **FileNameLength** is 0, the server SHOULD [<249>](#) set **Open.EnumerationSearchPattern** as "*" to search all entries.

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 [<250>](#) choose to support resuming enumerations by index number.

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**. [<251>](#) 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 [\[MS-CIFS\]](#) section 2.2.1.1.3. If SMB2_RETURN_SINGLE_ENTRY is set in the **Flags** field of the request, the server MUST return only 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.

After populating the buffer, the server MUST set **Open.EnumerationLocation** to the location of the next enumeration entry after the last one that was returned in the buffer. If there are no remaining entries, 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 zero 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 zero before querying the object store), the server MUST fail the request with STATUS_NO_MORE_FILES.

Otherwise, the server MUST construct an [SMB2 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.

The status code returned by this operation MUST be one of those defined in [\[MS-ERREF\]](#). Common status codes returned by this operation include:

- STATUS_SUCCESS
- STATUS_INSUFFICIENT_RESOURCES
- STATUS_ACCESS_DENIED
- STATUS_FILE_CLOSED

- STATUS_NETWORK_NAME_DELETED
- STATUS_USER_SESSION_DELETED
- STATUS_NETWORK_SESSION_EXPIRED
- STATUS_INVALID_PARAMETER
- STATUS_INVALID_INFO_CLASS
- STATUS_NO_SUCH_FILE
- STATUS_CANCELLED
- STATUS_NOT_SUPPORTED
- STATUS_OBJECT_NAME_INVALID
- STATUS_VOLUME_DISMOUNTED
- STATUS_INVALID_INFO_CLASS
- STATUS_FILE_CORRUPT_ERROR
- STATUS_NO_MORE_FILES

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 **GlobalSessionTable**. 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.

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. Otherwise, the server MUST locate the **Request** in **Connection.RequestList** for which **Request.MessageId** matches the **MessageId** value in the SMB2 header, and set **Request.Open** to the **Open**.

If **OutputBufferLength** is greater than **Connection.MaxTransactSize**[<252>](#), the server MUST fail the request with STATUS_INVALID_PARAMETER.

If **Connection.SupportsMultiCredit** is TRUE, the server MUST validate **CreditCharge** based on **OutputBufferLength**, as specified in section [3.3.5.2.4](#). If the validation fails, it MUST fail the request with STATUS_INVALID_PARAMETER.

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 [<253>](#) handle this operation asynchronously as specified in section [3.3.4.2](#).

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. [<254>](#) 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 indicates the maximum amount of notification data that it can accept by passing in the **OutputBufferLength** that is received from the client. An **OutputBufferLength** of zero indicates that the client allows the occurrence of an event but the client does not allow the notification data details. A Change notification request processed by the server with invalid bits in the **CompletionFilter** field MUST ignore the invalid bits and process the valid bits. If there are no valid bits in the **CompletionFilter**, the request will remain pending until the change notification is canceled or the directory handle is closed.

Change notification processing in the object store MUST be handled as specified in section [3.3.1.3](#). It is also outlined in [\[MS-CIFS\]](#) section 3.3.5.59.4.

If the server is unable to copy the results into the buffer of the [SMB2 CHANGE_NOTIFY Response](#), then the server MUST construct the response as described below, with an **OutputBufferLength** of zero, and set the Status in the SMB2 header to STATUS_NOTIFY_ENUM_DIR.

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.

The status code returned by this operation MUST be one of those defined in [\[MS-ERREF\]](#). Common status codes returned by this operation include:

- STATUS_SUCCESS
- STATUS_INSUFFICIENT_RESOURCES
- STATUS_ACCESS_DENIED
- STATUS_FILE_CLOSED
- STATUS_NETWORK_NAME_DELETED
- STATUS_USER_SESSION_DELETED

- STATUS_NETWORK_SESSION_EXPIRED
- STATUS_CANCELLED
- STATUS_INVALID_PARAMETER
- STATUS_NOTIFY_ENUM_DIR

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 **GlobalSessionTable**. 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.

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. Otherwise, the server MUST locate the **Request** in **Connection.RequestList** for which **Request.MessageId** matches the **MessageId** value in the SMB2 header, and set **Request.Open** to the **Open**.

If **OutputBufferLength** is greater than **Connection.MaxTransactSize**, the server MUST fail the request with STATUS_INVALID_PARAMETER.

If **Connection.SupportsMultiCredit** is TRUE, the server MUST validate **CreditCharge** based on the maximum of **InputBufferLength** and **OutputBufferLength**, as specified in section [3.3.5.2.4](#). If the validation fails, it MUST fail the request with STATUS_INVALID_PARAMETER.

The remaining processing for this request depends on the **InfoType** that is requested and described below.

The status code returned by this operation MUST be one of those defined in [\[MS-ERREF\]](#). Common status codes returned by this operation include:

- STATUS_SUCCESS
- STATUS_INSUFFICIENT_RESOURCES
- STATUS_ACCESS_DENIED
- STATUS_FILE_CLOSED
- STATUS_NETWORK_NAME_DELETED
- STATUS_USER_SESSION_DELETED
- STATUS_NETWORK_SESSION_EXPIRED

- STATUS_INVALID_PARAMETER
- STATUS_INVALID_INFO_CLASS
- STATUS_NOT_SUPPORTED
- STATUS_EA_LIST_INCONSISTENT
- STATUS_BUFFER_OVERFLOW
- STATUS_CANCELLED
- STATUS_INFO_LENGTH_MISMATCH

3.3.5.20.1 Handling SMB2_0_INFO_FILE

The information classes that are supported for querying files are listed in section [2.2.37](#). Documentation for these is provided in [\[MS-FSCC\]](#) section 2.4.

Requests for information classes that are not listed in section [2.2.37](#) but which are documented in section [2.4](#) of [\[MS-FSCC\]](#) SHOULD be failed with STATUS_NOT_SUPPORTED.

Requests for information classes not documented in [\[MS-FSCC\]](#) section 2.4 SHOULD [<255>](#) be failed with STATUS_INVALID_INFO_CLASS.

If the request is for the FilePositionInformation information class, the SMB2 server SHOULD [<256>](#) set the **CurrentByteOffset** field to zero. The **CurrentByteOffset** field is part of the **FILE_POSITION_INFORMATION** structure specified in section [2.4.32](#) of [\[MS-FSCC\]](#).

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. [<257>](#) 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 either zero or 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 zero.

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.

FullEaList: The list of extended attribute entries maintained by underlying object store.

EaIndex: Index of the EA in **FullEaList** to start enumerating EA entries. It starts from 1.

EaList: The list of FILE_GET_EA_INFORMATION structures as specified in [\[MS-FSCC\]](#) section 2.4.15.1.

If the object store supports security and the information class is set to FileFullEaInformation, the server MUST return one or more extended attribute information associated to the current Open, as follows:

- If **EaList** is specified by the client, the server MUST query the EA entries from **FullEaList** through the EA names in **EaList** until the buffer is full or has run to the end of **EaList**. The **EaList** is contained at the offset **InputBufferOffset**, starting from the SMB2 header with the length set to **InputBufferLength**.
- If SL_INDEX_SPECIFIED is not set in the **Flags** field and **EaList** is not specified, the server MUST enumerate the EA entries from **FullEaList** starting at **Open.CurrentEaIndex** until the buffer is full or has run out of the EA entries in **FullEaList**. **Open.CurrentEaIndex** MUST be incremented by the number of EA entries returned to the client.
- If SL_RESTART_SCAN is set in the **Flags** field, the server MUST ignore it if either SL_INDEX_SPECIFIED is set in the **Flags** field or **EaList** is specified by the client. Otherwise, the server MUST set **Open.CurrentEaIndex** to 1.
- If SL_INDEX_SPECIFIED is set in the **Flags** field, it SHOULD be ignored by the server if **EaList** is specified by the client. Otherwise, the server MUST use **EaIndex** as the starting index in **FullEaList** to enumerate the EA entries until the buffer is full or has run out of the EA entries in **FullEaList**. If an out-of-range **EaIndex** is specified, the server MUST fail the request with STATUS_NONEXISTENT_EA_ENTRY.
- If SL_RETURN_SINGLE_ENTRY is set in the **Flags** field, the server MUST return the single EA entry 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 section [2.2.37](#). Documentation for these is provided in [\[MS-FSCC\]](#) section 2.5.

Requests for information classes not listed in section [2.2.37](#) but documented in [\[MS-FSCC\]](#) section 2.5 SHOULD be failed with STATUS_NOT_SUPPORTED.

Requests for information classes not documented in [\[MS-FSCC\]](#) section 2.5 SHOULD 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. [<258>](#) 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 either zero or 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 [<259>](#) with **ByteCount** set to zero.

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. [<259>](#)

3.3.5.20.3 Handling SMB2_0_INFO_SECURITY

The following section assumes knowledge about security concepts, as described in [\[MS-WSQ\]](#) section 3.1.2 and specified in [\[MS-DTYP\]](#).

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 **AdditionalInformation** 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 **AdditionalInformation** 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. [<260>](#)

The fields required in the resulting security descriptor are denoted by the flags given in the **AdditionalInformation** field of the request.

If the **OutputBufferLength** given in the client request is either zero or is insufficient to hold the information requested, the server MUST fail the request with STATUS_BUFFER_TOO_SMALL. The server MUST return error data containing the buffer size, in bytes, that would be required to return the requested information, as specified in section 2.2.2 with ByteCount set to 4. 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 2.4.2.2. <261>

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 retrieved from the underlying volume that hosts the open in the object store. <262>

FullQuotaList: The list of the volume's quota information entries maintained by the underlying object store.

SidList: The list of **FILE_GET_QUOTA_INFORMATION** structures as specified in [\[MS-FSCC\]](#) section 2.4.33.1.

- If **ReturnSingle** is TRUE, the server MUST return at most a single quota information entry to the client.
- If **SidListLength** is nonzero, the server MUST ignore the values of **StartSidOffset** and **StartSidLength**, and enumerate the quota information entries for all the SIDs specified in **SidList**. If **SidList** is not a list of **FILE_GET_QUOTA_INFORMATION** structures linked via the **NextEntryOffset** field, the server MUST fail the request with STATUS_INVALID_PARAMETER. If the server can't find the corresponding quota information entry through the SID specified in the **FILE_GET_QUOTA_INFORMATION** structure, then the server MUST return **FILE_GET_QUOTA_INFORMATION** for the SID with the following fields set to zero: **ChangeTime**, **QuotaUsed**, **QuotaThreshold**, and **QuotaLimit**.
- If **SidListLength** is zero, and **StartSidLength** or **StartSidOffset** is nonzero, the server SHOULD fail the request with STATUS_INVALID_PARAMETER.

- If **StartSidLength** or **StartSidOffset** or **SidListLength** are nonzero, the server MUST ignore the value of **RestartScan**.
- If **StartSidLength** and **StartSidOffset** and **SidListLength** are all zero, the server MUST check the value of **RestartScan**. If **RestartScan** is TRUE, the server MUST set **Open.CurrentQuotaIndex** to 1. The server MUST use **Open.CurrentQuotaIndex** as the starting index in **FullQuotaList** to enumerate the quota information entries until the buffer is full or has run out of the quota information entries in **FullQuotaList**. **Open.CurrentQuotaIndex** MUST be incremented by the number of quota information entries returned to the client.

The server MUST return STATUS_SUCCESS if at least one FILE_QUOTA_INFORMATION entry is returned.

If the **OutputBufferLength** given in the client request is either zero or is insufficient to hold single FILE_QUOTA_INFORMATION entry, the server MUST fail the request with STATUS_BUFFER_TOO_SMALL and return error data, as specified in section [2.2.2](#), with **ByteCount** set to zero.

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 **GlobalSessionTable**. 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.

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. Otherwise, the server MUST locate the **Request** in **Connection.RequestList** for which **Request.MessageId** matches the **MessageId** value in the SMB2 header, and set **Request.Open** to the **Open**.

If **BufferLength** is greater than **Connection.MaxTransactSize**, the server MUST fail the request with STATUS_INVALID_PARAMETER.

If **Connection.SupportsMultiCredit** is TRUE, the server MUST validate **CreditCharge** based on **BufferLength**, as specified in section [3.3.5.2.4](#). If the validation fails, it MUST fail the request with STATUS_INVALID_PARAMETER.

The remaining processing for this request depends on the **InfoType** requested, as described below.

The status code returned by this operation MUST be one of those defined in [\[MS-ERREF\]](#). Common status codes returned by this operation include:

- STATUS_SUCCESS
- STATUS_INSUFFICIENT_RESOURCES
- STATUS_ACCESS_DENIED
- STATUS_FILE_CLOSED
- STATUS_NETWORK_NAME_DELETED
- STATUS_USER_SESSION_DELETED
- STATUS_NETWORK_SESSION_EXPIRED
- STATUS_INVALID_PARAMETER
- STATUS_INVALID_INFO_CLASS
- STATUS_NOT_SUPPORTED
- STATUS_EA_LIST_INCONSISTENT
- STATUS_CANCELLED

3.3.5.21.1 Handling SMB2_0_INFO_FILE

The information classes that are supported for setting file information are listed in section [2.2.39](#). Documentation for these is provided in [\[MS-FSCC\]](#) section 2.4.

Requests for information classes documented in [\[MS-FSCC\]](#) section 2.4 with "Set" not specified in the Uses column are not allowed and SHOULD be failed with STATUS_INVALID_INFO_CLASS.

Requests for information classes not documented in section [2.4](#) of [MS-FSCC] SHOULD [<263>](#) be failed with STATUS_INVALID_INFO_CLASS.

Requests for information classes not listed in section [2.2.39](#) but documented in [\[MS-FSCC\]](#) section 2.4 with "Set" specified in the Uses column are not allowed and SHOULD be failed with STATUS_NOT_SUPPORTED.

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 FileFullEaInformation and the EA buffer in the **Buffer** field is not in a valid format, the server MUST fail the request with STATUS_EA_LIST_INCONSISTENT.

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. <264> 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](#).

If the underlying object store returns successfully, the information class is FileRenameInformation, the server implements SMB 2.1 and supports leasing, and **Open.Lease** is not NULL, the server MUST update **Open.Lease.FileName** to the new name for the file.

The response MUST then be sent to the client.

3.3.5.21.2 Handling SMB2_0_INFO_FILESYSTEM

The information classes that are supported for setting underlying object store information are listed in section [2.2.39](#). Documentation for these is provided [\[MS-FSCC\]](#) section 2.5. Requests for information classes not listed in section [2.2.39](#) but documented in section [2.5](#) of [\[MS-FSCC\]](#) for Uses of "Set" or "LOCAL" MUST be failed with STATUS_NOT_SUPPORTED. Requests for information classes not documented in section [2.5](#) of [\[MS-FSCC\]](#) or documented in section [2.5](#) of [\[MS-FSCC\]](#) for Uses of only "Query" MUST be failed with STATUS_INVALID_INFO_CLASS.

If the object store supports security and the information class is FileFsControlInformation or FileFsObjectIdInformation 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. <265> 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.

3.3.5.21.3 Handling SMB2_0_INFO_SECURITY

The following section assumes knowledge about security concepts as described in [\[MS-WSO\]](#) section 3.1.2 and specified in [\[MS-DTYP\]](#). <266>

1. If SACL_SECURITY_INFORMATION is set in the **AdditionalInformation** 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 **AdditionalInformation** 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 **AdditionalInformation** 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. [<267>](#267)

The fields being applied in the provided security descriptor are denoted by the flags given in the **AdditionalInformation** 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](#SMB2_SET_INFO_Response) following the syntax given in section [2.2.40](#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 [\[MS-DTYP\]](#MS-DTYP) section 2.4.2.2. [<268>](#268)

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. [<269>](#269)

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](#SMB2_SET_INFO_Response) following the syntax given in section [2.2.40](#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](#SMB2_header) with a **Command** value equal to SMB2_OPLOCK_BREAK, message handling proceeds as follows:

- If **Connection.Dialect** is equal to "2.100" and the **StructureSize** of the request is equal to 36, the server MUST process the request as described in section [3.3.5.22.2](#3.3.5.22.2).

- Otherwise, the server MUST process the request as described in section [3.3.5.22.1](#).

3.3.5.22.1 Processing an Oplock Acknowledgment

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 **GlobalSessionTable**. 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 InProgress, the server MUST fail the request with STATUS_ACCESS_DENIED.

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. Otherwise, the server MUST locate the **Request** in **Connection.RequestList** for which **Request.MessageId** matches the **MessageId** value in the SMB2 header, and set **Request.Open** to the **Open**.

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 SMB2_OPLOCK_LEVEL_EXCLUSIVE or SMB2_OPLOCK_LEVEL_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 [3.3.4.6](#). 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 value:

- **OplockLevel** MUST be set to **Open.OplockLevel**.

This response MUST then be sent to the client.

The status code returned by this operation MUST be one of those defined in [\[MS-ERREF\]](#). Common status codes returned by this operation include:

- STATUS_ACCESS_DENIED
- STATUS_FILE_CLOSED
- STATUS_INVALID_OPLOCK_PROTOCOL
- STATUS_INVALID_PARAMETER
- STATUS_INVALID_DEVICE_STATE

- STATUS_NETWORK_NAME_DELETED
- STATUS_USER_SESSION_DELETED

3.3.5.22.2 Processing a Lease Acknowledgment

The server MUST locate the session that is taking the action by using the **SessionId** in the **SMB2 header** of the request to do a lookup on the **GlobalSessionTable**. 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 InProgress, the server MUST fail the request with STATUS_ACCESS_DENIED.

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 Lease Table by performing a lookup in **GlobalLeaseTableList** using **Connection.ClientGuid** as the lookup key. If no lease table is found, the server MUST fail the request with STATUS_FILE_CLOSED.

The server MUST locate the lease on which the client is acknowledging a lease break by performing a lookup in **LeaseTable.LeaseList** using the **LeaseKey** of the request as the lookup key. If no lease is found, the server MUST fail the request with STATUS_FILE_CLOSED.

If **Lease.Breaking** is FALSE, the server MUST fail the request with STATUS_UNSUCCESSFUL.

If **LeaseState** is not \leq **Lease.BreakToLeaseState**, the server MUST fail the request with STATUS_REQUEST_NOT_ACCEPTED.

The server completes the lease break request received from the object store as described in section [3.3.4.7](#). The server MUST set **Lease.LeaseState** to **LeaseState** received in the request, and MUST set **Lease.Breaking** to FALSE.

The server then MUST construct an oplock break response using the syntax specified in section [2.2.25](#) with the following values:

- **LeaseKey** MUST be set to **Lease.LeaseKey**.
- **LeaseState** MUST be set to **Lease.LeaseState**.

This response MUST then be sent to the client.

The status code returned by this operation MUST be one of those defined in [\[MS-ERREF\]](#). Common status codes returned by this operation include:

- STATUS_ACCESS_DENIED
- STATUS_FILE_CLOSED
- STATUS_INVALID_OPLOCK_PROTOCOL
- STATUS_INVALID_PARAMETER
- STATUS_INVALID_DEVICE_STATE
- STATUS_NETWORK_NAME_DELETED
- STATUS_USER_SESSION_DELETED

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 or lease 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.

If the server implements SMB 2.1 and supports leasing, the server MUST scan for lease breaks that have not been acknowledged by the client within the configured time. It does this by enumerating all lease tables in **GlobalLeaseTableList**. For each lease table, it enumerates all leases in **LeaseTable.LeaseList**. For each lease, if **Lease.Breaking** is TRUE and **Lease.LeaseBreakTimeout** is earlier than the current time, the server MUST acknowledge the lease break to the underlying object store represented by the opens in **Lease.LeaseOpens**, and set **Lease.LeaseState** 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, and no leases (if implemented) have **Lease.Breaking** set to TRUE, the timer MUST NOT be restarted.

3.3.6.2 Durable Open Scavenger Timer Event

The durable open scavenger timer MUST be started (if it is not already active) when the transport connection associated with a durable open is lost.

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.IsDurable** is TRUE, **Open.Connection** is NULL, and **Open.DurableOpenTimeout** is earlier than the current time, the server MUST close the open as specified in section [3.3.4.17](#).

The timer MUST then be restarted to expire again at the time of the next durable open time-out. If no other durable 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 **GlobalSessionTable**. If the **Session.State** is Valid and the **Session.ExpirationTime** has passed, the **Session.State** MUST be set to Expired and **ServerStatistics.sts0_stimedout** MUST be increased by 1.

3.3.6.4 Resilient Open Scavenger Timer Event

If the server implements SMB 2.1 and supports resiliency, it MUST implement this timer event.

When the resilient open scavenger timer expires, the server MUST scan for resilient 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.IsResilient** is TRUE, **Open.Connection** is NULL

and **Open.ResilientOpenTimeout** is earlier than the current time, the server MUST close the Open as specified in section [3.3.4.17](#).

The timer MUST then be restarted to expire again at the time of the next resilient open time-out and **ResilientOpenScavengerExpiryTime** MUST be set to the next resilient open time-out. If no other resilient opens have **Open.Connection** equal to NULL, the timer MUST NOT be restarted.

3.3.7 Other Local Events

3.3.7.1 Handling Loss of a Connection

When the underlying transport indicates loss of a connection or after the server initiates a transport disconnect, the server MUST enumerate all sessions established over the connection in **GlobalSessionTable** and perform the following actions for each session:

- The server MUST iterate over the **Session.OpenTable** and determine whether each **Open** is to be preserved for reconnect. If any of the following conditions is satisfied, it indicates that the **Open** is to be preserved for reconnect.
 - The server implements SMB 2.1 and supports leasing and **Open.IsResilient** is TRUE.
 - **Open.OplockLevel** is equal to SMB2_OPLOCK_LEVEL_BATCH and **Open.OplockState** is equal to Held, and **Open.IsDurable** is TRUE.
 - **Open.OplockLevel** is equal to SMB2_OPLOCK_LEVEL_LEASE, **Lease.LeaseState** contains SMB2_LEASE_HANDLE_CACHING, **Open.OplockState** is equal to Held, and **Open.IsDurable** is TRUE.

If the **Open** is to be preserved for reconnect, perform the following actions:

- Set **Open.Connection** to NULL, **Open.Session** to NULL, **Open.TreeConnect** to NULL.
- If **Open.IsResilient** is TRUE, set **Open.ResilientOpenTimeout** to the current time plus **Open.ResiliencyTimeout**. The server MUST start or reset the Resilient Open Scavenger Timer, as specified in section [3.3.2.4](#), under the following conditions:
 - If the Resilient Open Scavenger Timer is not already active.
 - If the Resilient Open Scavenger Timer is active and **ResilientOpenScavengerExpiryTime** is greater than **Open.ResilientOpenTimeout**.

In both of the preceding cases, the server MUST set the timer to expire at **Open.ResilientOpenTimeout** and MUST set **ResilientOpenScavengerExpiryTime** to **Open.ResilientOpenTimeout**.

- If **Open.IsDurable** is TRUE, set **Open.DurableOpenTimeout** to the current time plus an implementation-specific [<270>](#) default value.

If the **Open** is not to be preserved for reconnect, the server MUST close the Open as specified in section [3.3.4.17](#).

- The server MUST disconnect every **TreeConnect** in **Session.TreeConnectTable** and deregister the **TreeConnect** by invoking the event specified in [\[MS-SRVS\]](#) section 3.1.6.7, providing the tuple **<TreeConnect.Share.ServerName, TreeConnect.Share.Name>** and **TreeConnect.TreeGlobalId** as the input parameters, and the **TreeConnect** MUST be removed from **Session.TreeConnectTable** and freed. For each deregistered **TreeConnect**, **TreeConnect.Share.CurrentUses** MUST be decreased by 1.

- The server MUST deregister the **Session** by invoking the event specified in [\[MS-SRVS\]](#) section 3.1.6.3, providing **Session.SessionGlobalId** as the input parameter, and the **Session** MUST be removed from **GlobalSessionTable** and freed. **ServerStatistics.sts0_sopens** MUST be decreased by 1.

All requests in **Connection.RequestList** MUST be canceled. The server SHOULD [<271>](#) pass the **CancelRequestId** to the object store to request cancellation of the pending operation.

The server MUST invoke the event specified in [\[MS-SRVS\]](#) section 3.1.6.16 to update the connection count by providing the tuple **<Connection.TransportName,FALSE>**.

The connection MUST be removed from **ConnectionList** and MUST be freed.

If any opens were preserved for reconnect, the server MUST start the durable open scavenger timer and resilient open scavenger timer if necessary, as specified in sections [3.3.6.2](#) and [3.3.6.4](#).

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 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.

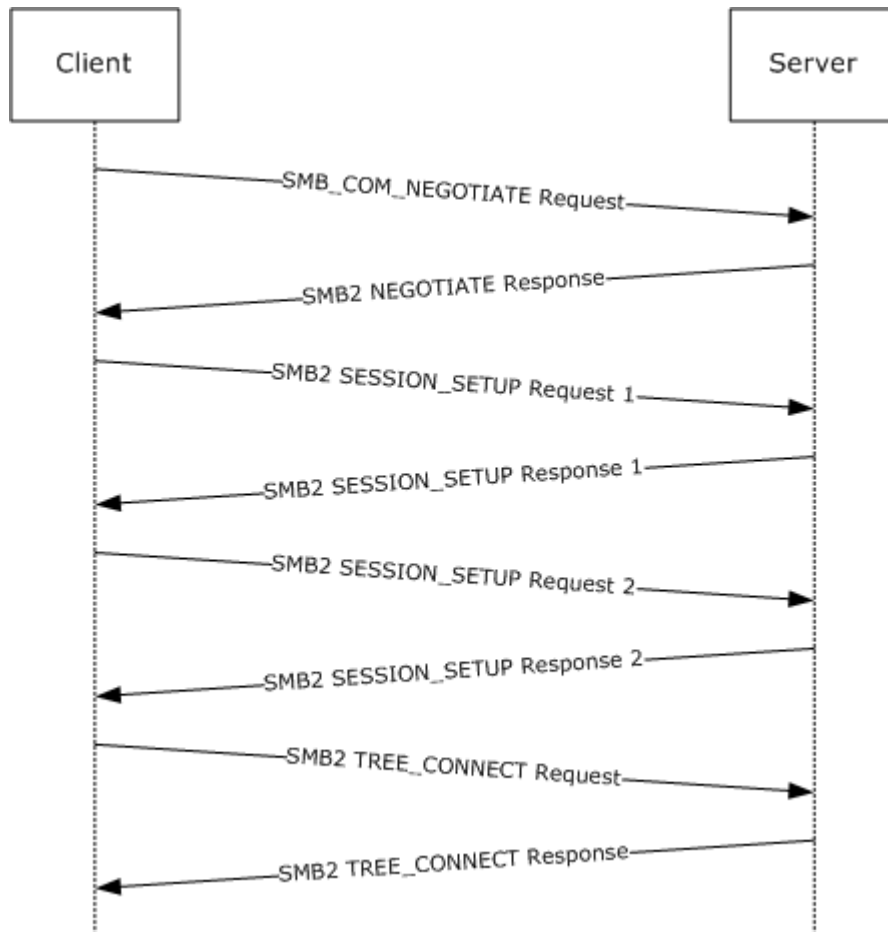


Figure 5: Client negotiating SMB2 with SMB-style negotiate

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: 0xFFFF, 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_KNOWN_LONG_NAMES: May return long file names
 Bit01: (.....1.) SMB_FLAGS2_KNOWN_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 (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 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 (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, **CreditResponse** of 5, **Status** equal to STATUS_SUCCESS, **SessionId** of 0x40000000009, 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 Negotiating SMB 2.10 dialect by using Multi-Protocol Negotiate

The following diagram shows the steps taken by a client that is negotiating SMB 2.10 dialect by using an SMB-style negotiate.

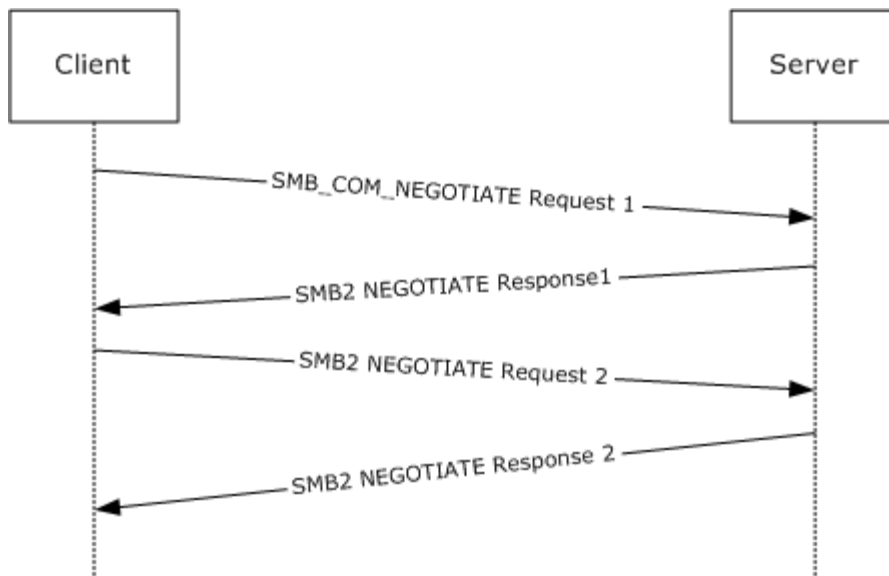


Figure 6: Client negotiating SMB 2.10 dialect with SMB-style negotiate

1. The client sends an SMB negotiate packet with the string "SMB 2.???" 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, SMB 2.???
Protocol: SMB
Command: Negotiate 114(0x72)
NTStatus: 0x0, Facility = FACILITY_SYSTEM, Severity = STATUS_SEVERITY_SUCCESS, Code =
(0) STATUS_SUCCESS
Code: (.....0000000000000000) (0) STATUS_SUCCESS
Facility: (...0000000000000000) FACILITY_SYSTEM
Customer: (...0.....) NOT Customer Defined
Severity: (00.....) STATUS_SEVERITY_SUCCESS
SMBHeader: Command, TID: 0xFFFF, PID: 0xFEFF, UID: 0x0000, MID: 0x0000
Flags: 24 (0x18)
LockAndRead: (.....0) LOCK_AND_READ and WRITE_AND_UNLOCK NOT supported
(Obsolete) (SMB_FLAGS_LOCK_AND_READ_OK)
NoAck: (.....0..) An ACK response is needed (SMB_FLAGS_SEND_NO_ACK[only
applicable when SMB transport is NetBIOS over IPX])
Reserved_bit2: (.....0..) Reserved (Must Be Zero)
CaseInsensitive: (...1...) SMB paths are caseinsensitive (SMB_FLAGS_CASE_INSENSITIVE)
Canonicalized: (...1....) Canonicalized File and pathnames (Obsolete)
(SMB_FLAGS_CANONICALIZED_PATHS)
Oplock: (...0.....) Oplocks NOT supported for OPEN, CREATE & CREATE_NEW
(Obsolete) (SMB_FLAGS_OPLOCK)
OplockNotify: (.0.....) Notifications NOT supported for OPEN, CREATE & CREATE_NEW
(Obsolete) (SMB_FLAGS_OPLOCK_NOTIFY_ANY)
FromServer: (0.....) Command SMB is being sent from the client
(SMB_FLAGS_SERVER_TO_REDIR)
Flags2: 51283 (0xC853)
KnowsLongFiles: (.....1) Understands Long File Names
(SMB_FLAGS2_KNOWS_LONG_NAMES)
ExtendedAttribs: (.....1.) Understands extended attributes
(SMB_FLAGS2_KNOWS_EAS)
  
```



```

SignEnabled:      (.....0..) Security signatures NOT enabled
(SMB_FLAGS2_SMB_SECURITY_SIGNATURE)
Compressed:      (.....0...) Compression Disabled for REQ_NT_WRITE_ANDX and
RESP_READ_ANDX (SMB_FLAGS2_COMPRESSED)
SignRequired:    (.....1....) Security Signatures are required
(SMB_FLAGS2_SMB_SECURITY_SIGNATURE_REQUIRED)
Reserved_bit5:   (.....0.....) Reserved (Must Be Zero)
LongFileNames:   (.....1.....) Use Long File Names (SMB_FLAGS2_IS_LONG_NAME)
Reserved_bits7_9: (.....000.....) Reserved (Must Be Zero)
ReparsePath:     (.....0.....) NOT a Reparse path (SMB_FLAGS2_REPARSE_PATH)
ExtSecurity:     (....1.....) Aware of extended security
(SMB_FLAGS2_EXTENDED_SECURITY)
Dfs:             (...0.....) NO DFS namespace (SMB_FLAGS2_DFS)
Paging:          (...0.....) Read operation will NOT be permitted unless user
has permission (NO Paging IO) (SMB_FLAGS2_PAGING_IO)
StatusCodes:     (.1.....) Using 32bit NT status error codes
(SMB_FLAGS2_NT_STATUS)
Unicode:         (1.....) Using UNICODE strings (SMB_FLAGS2_UNICODE)
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: 120 (0x78)
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
Dialect: SMB 2.???
BufferFormat: Dialect 2(0x2)
DialectName: SMB 2.???

```

2. The server receives the SMB negotiate request and finds the "SMB 2.???" string in the dialect string list. The server responds with an [SMB2 NEGOTIATE Response](#) with the DialectRevision set to 0x02ff.

```

Smb2: R NEGOTIATE (0x0), GUID={1ED9580F5FEF1AA04B9DDB1C77C63757}, Mid = 0
SMBIdentifier: SMB
SMB2Header: R NEGOTIATE (0x0)
Size: 64 (0x40)
Epoch: 0 (0x0)
Status: 0x0, Facility = FACILITY_SYSTEM, Severity = STATUS_SEVERITY_SUCCESS, Code =
(0) STATUS_SUCCESS
Code: (.....0000000000000000) (0) STATUS_SUCCESS
Facility: (...000000000000.....) FACILITY_SYSTEM
Customer: (...0.....) NOT Customer Defined
Severity: (00.....) STATUS_SEVERITY_SUCCESS
Command: NEGOTIATE (0x0)
Credits: 1 (0x1)
Flags: 0x1
ServerToRedir: (.....1) Server to Client
(SMB2_FLAGS_SERVER_TO_REDIR)
AsyncCommand: (.....0.) Command is not asynchronous
(SMB2_FLAGS_ASYNC_COMMAND)
Related: (.....0..) Packet is single message
(SMB2_FLAGS_RELATED_OPERATIONS)
Signed: (.....0...) Packet is not signed
(SMB2_FLAGS_SIGNED)
Reserved4_27: (...00000000000000000000000000000000....)
DFS: (...0.....) Command is not a DFS Operation
(SMB2_FLAGS_DFS_OPERATIONS)
Reserved29_31: (000.....)
NextCommand: 0 (0x0)
MessageId: 0 (0x0)
ProcessId: 0 (0x0)
TreeId: 0 (0x0)
SessionId: 0 (0x0)
Signature: Binary Large Object (16 Bytes)
RNegotiate:
Size: 65 (0x41)
SecurityMode: Signing Enabled (0x1)
DialectRevision: 767 (0x2FF)
Reserved: 0 (0x0)
Guid: {1ED9580F5FEF1AA04B9DDB1C77C63757}
Capabilities: 0x3
DFS: (.....1) DFS available
Reserved_bits1_31: (00000000000000000000000000000001.) Reserved
MaxTransactSize: 1048576 (0x100000)
MaxReadSize: 1048576 (0x100000)
MaxWriteSize: 1048576 (0x100000)
SystemTime: 12/29/2008, 11:18:59 PM
SystemStartTime: 12/05/2008, 11:55:51 PM
SecurityBufferOffset: 128 (0x80)
SecurityBufferLength: 120 (0x78)
Reserved2: 541936672 (0x204D4C20)
securityBlob:

```

3. The client receives the SMB2 NEGOTIATE Response. The client issues a new [SMB2 NEGOTIATE Request](#) with a new dialect 0x0210 appended along with other SMB2 dialects.

```

Smb2: C NEGOTIATE (0x0), GUID={9879BE56-0D00-58BA-11DD-D5F0AF3A5B5D}, Mid = 1

```

4. The server receives the SMB2 negotiate request and finds dialect 0x0210. The server sends an SMB2 NEGOTIATE Response with DialectRevision set to 0x0210.

275 / 341

```
Facility: (...000000000000...) FACILITY_SYSTEM
Customer: (...0...) NOT Customer Defined
Severity: (00...) STATUS_SEVERITY_SUCCESS
Command: NEGOTIATE (0x0)
Credits: 1 (0x1)
Flags: 0x1
ServerToRedir: (...1) Server to Client
(SMB2_FLAGS_SERVER_TO_REDIR)
AsyncCommand: (...0.) Command is not asynchronous
(SMB2_FLAGS_ASYNC_COMMAND)
Related: (...0..) Packet is single message
(SMB2_FLAGS_RELATED_OPERATIONS)
Signed: (...0...) Packet is not signed
(SMB2_FLAGS_SIGNED)
Reserved4_27: (...00000000000000000000000000000000....)
DFS: (...0...) Command is not a DFS Operation
(SMB2_FLAGS_DFS_OPERATIONS)
Reserved29_31: (000...)
NextCommand: 0 (0x0)
MessageId: 1 (0x1)
ProcessId: 65279 (0xFEFF)
TreeId: 0 (0x0)
SessionId: 0 (0x0)
Signature: Binary Large Object (16 Bytes)
RNegotiate:
Size: 65 (0x41)
SecurityMode: Signing Enabled (0x1)
DialectRevision: 528 (0x210)
Reserved: 0 (0x0)
Guid: {1ED9580F-5FEF-1AA0-4B9D-DB1C77C63757}
Capabilities: 0x3
DFS: (...1) DFS available
Reserved_bits1_31: (00000000000000000000000000000001.) Reserved
MaxTransactSize: 1048576 (0x100000)
MaxReadSize: 1048576 (0x100000)
MaxWriteSize: 1048576 (0x100000)
SystemTime: 12/29/2008, 11:18:59 PM
SystemStartTime: 12/05/2008, 11:55:51 PM
SecurityBufferOffset: 128 (0x80)
SecurityBufferLength: 120 (0x78)
Reserved2: 0 (0x0)
securityBlob:
```

4.3 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.

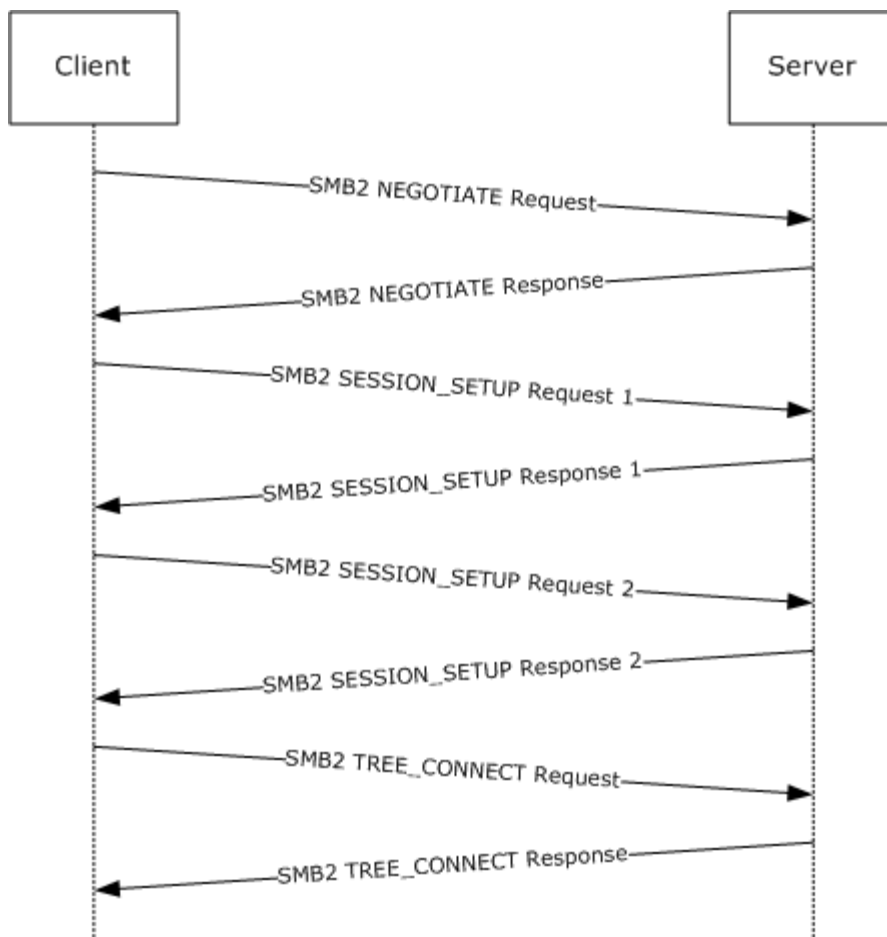


Figure 7: Client negotiating SMB2 with SMB2 negotiate

1. The client sends an SMB2 negotiate packet with the dialect 0x0202 in the **Dialects** 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: 514 (0x0202)

```

2. The server receives the [SMB2 NEGOTIATE Request](#) and finds dialect 0x0202. 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, **CreditResponse** of 5, **Status** equal to STATUS_SUCCESS, **SessionId** of 0x40000000009, 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.

4.4 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.

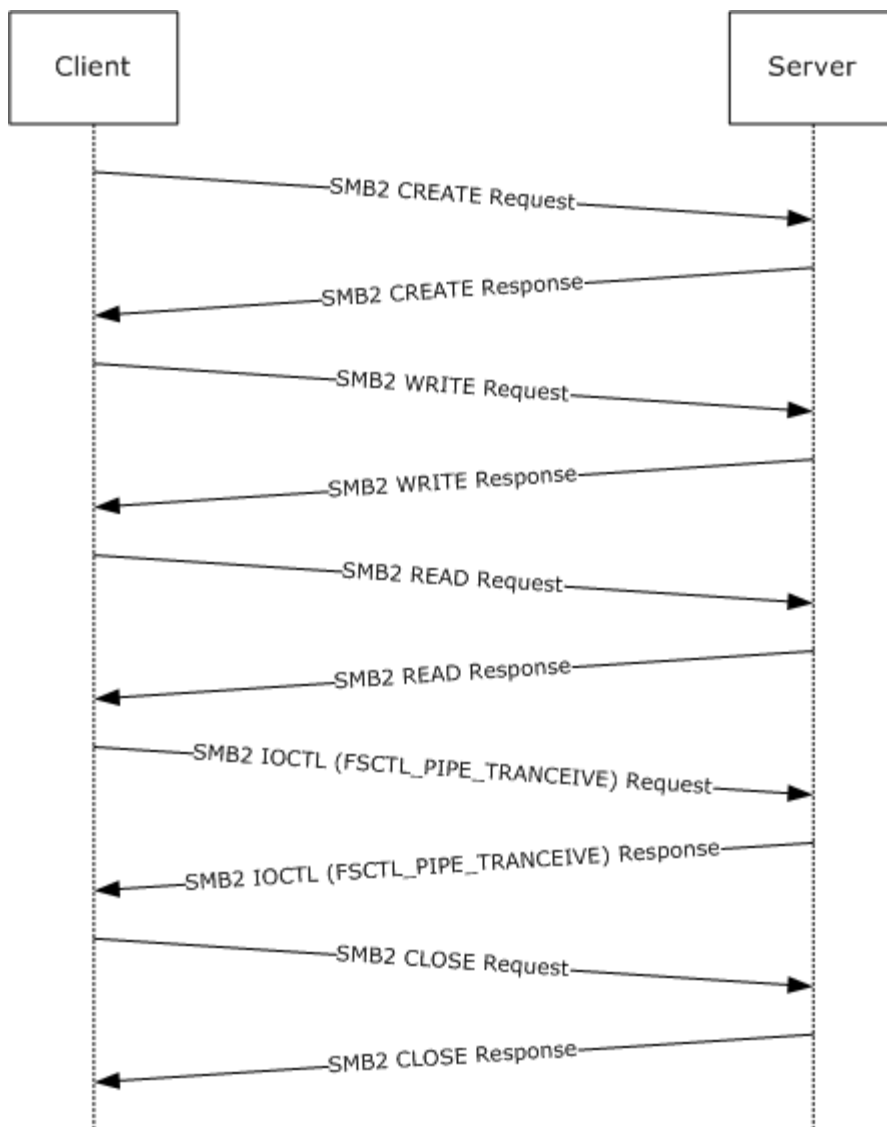


Figure 8: Executing an operation on a named pipe

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 (0x4000000000D)
CCreate:
Size: 57 (0x39)
SecurityFlags: 0 (0x0)
RequestedOplockLevel: 9 (0x9)
ImpersonationLevel: 2 (0x2)
SmbCreateFlags: 0 (0x0)
Reserved: 0(0x0)
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 (0x4000000000D)
RCreate:
Size: 89 (0x59)
OplockLevel: 0 (0x0)
Reserved1: 9 (0x9)
CreateAction: 1 (0x1)
CreationTime: 0 (0x0)
LastAccessTime: 0 (0x0)
LastWriteTime: 0 (0x0)
ChangeTime: 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 (0x4000000000D)
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 (0x4000000000D)
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)
CtIoctl:
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.

```

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)
R_IOCTL:
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 (0x4000000000D)
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 (0x4000000000D)
RClose:
```

```

Size: 60 (0x3C)
Flags: 0 (0x0)
Reserved: 0 (0x0)
CreationTime: 0 (0x0)
LastAccessTime: 0 (0x0)
LastWriteTime: 0 (0x0)
ChangeTime: 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.....)

```

4.5 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.

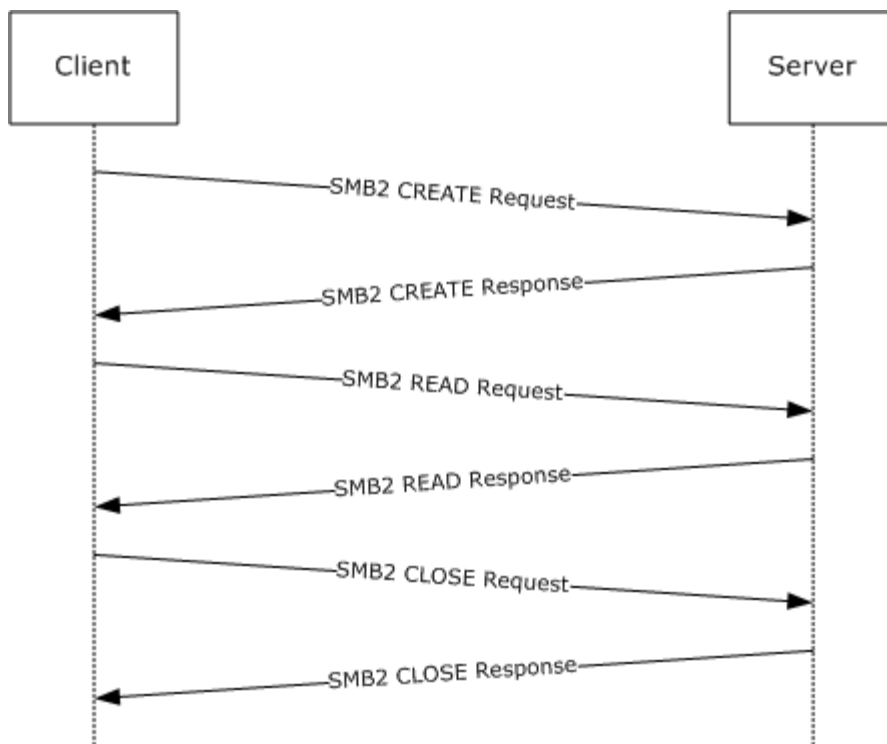


Figure 9: Reading from a remote file

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)
Reserved: 0 (0x0)
  
```

```

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)
ChangeTime: 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)
ChangeTime: 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

```

4.6 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 0x40000000015 for all requests and responses listed below.

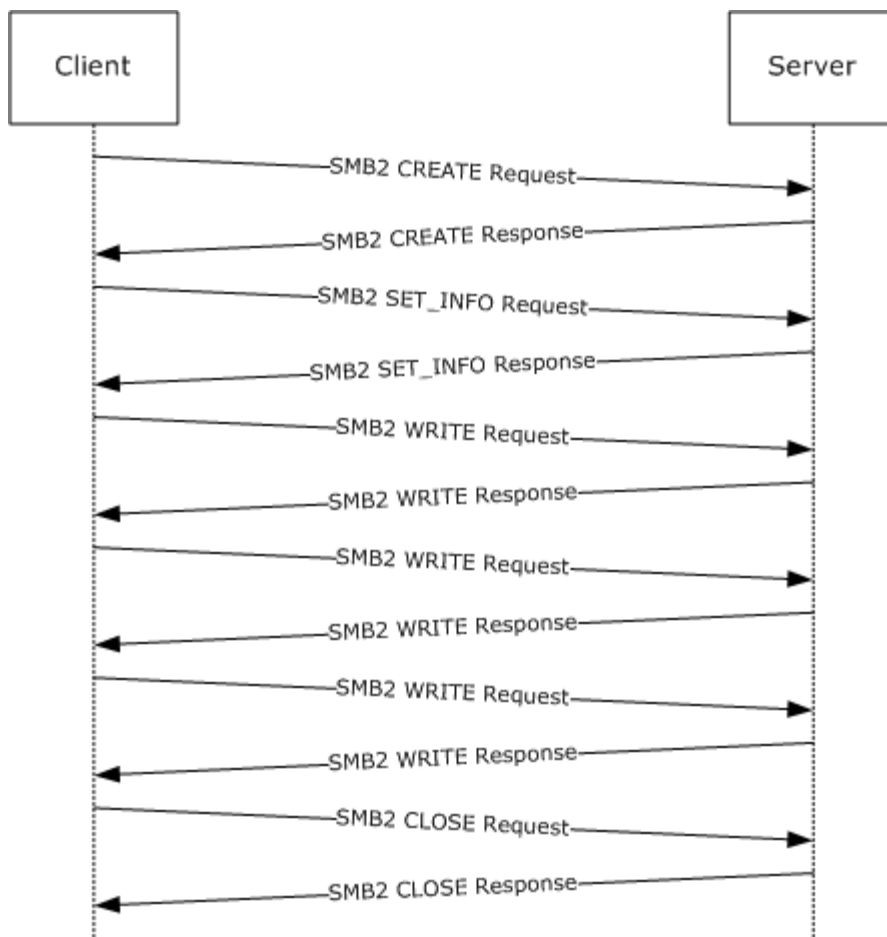


Figure 10: Writing to a remote file

1. The client sends an [SMB2 CREATE Request](#) 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)
ServerToRedirect: .....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 (0x400000000015)
  
```

```

CCreate:
Size: 57 (0x39)
SecurityFlags: 0 (0x0)
RequestedOplockLevel: 9 (0x9)
ImpersonationLevel: 2 (0x2)
SmbCreateFlags: 0 (0x0)
Reserved: 0 (0x0)
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)
ChangeTime: 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** (specified in [MS-FSCC](#) section 2.4.13) 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 (0x400000000015)
CSetInfo:
Size: 33 (0x21)
InfoType: 1 (0x1)
FileInformationClass:
    FileEndOfFileInformation
BufferLength: 8 (0x8)
BufferOffset: 96 (0x60)
Reserved: 0 (0x0)
AdditionalInformation: 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 (0x400000000015)
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)
ChangeTime: 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

```

4.7 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.

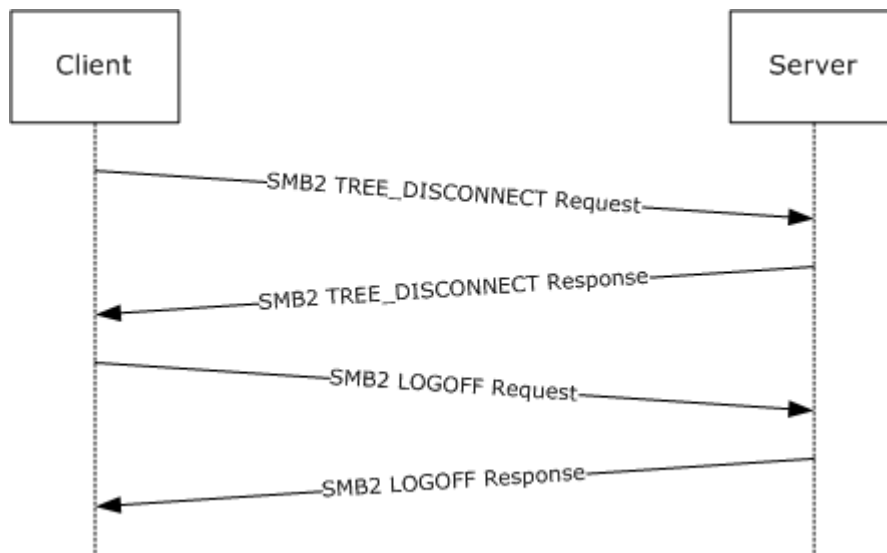


Figure 11: Disconnecting a share and logging off a session

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 Request](#) 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)
CLogout:
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)
RLogout:
Size: 4 (0x4)
Reserved: 0 (0x0)

```

5 Security

The following sections specify security considerations for implementers of the Server Message Block (SMB) Version 2 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 protocol does not require cancel requests from the client to the server to be signed if message signing is enabled. This could allow attackers to cancel previously sent messages from the client to the server on the same SMB2 transport connection.

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: Product Behavior

The information in this specification is applicable to the following Microsoft products or supplemental software. References to product versions include released service packs:

- Windows Vista® operating system
- Windows Server® 2008 operating system
- Windows® 7 operating system
- Windows Server® 2008 R2 operating system

Exceptions, if any, are noted below. If a service pack or Quick Fix Engineering (QFE) number appears with the product version, behavior changed in that service pack or QFE. The new behavior also applies to subsequent service packs of the product unless otherwise specified. If a product edition appears with the product version, behavior is different in that product edition.

Unless otherwise specified, any statement of optional behavior in this specification that is prescribed using the terms SHOULD or SHOULD NOT implies product behavior in accordance with the SHOULD or SHOULD NOT prescription. Unless otherwise specified, the term MAY implies that the product does not follow the prescription.

[<1> Section 1.6:](#) The Server Message Block (SMB) Version 2.002 Protocol is available only in Windows Vista, Windows Server 2008, Windows 7, and Windows Server 2008 R2. SMB Version 2.1 is available only in Windows 7 and Windows Server 2008 R2. Windows Vista, Windows Server 2008, Windows 7 and Windows Server 2008 R2 default to using the highest common dialect of the SMB 2 Protocol when it is available. Windows 95, Windows 98, Windows Millennium Edition, Windows NT 4.0, Windows 2000, Windows XP, and Windows Server 2003, support the SMB Protocol, as specified in [\[MS-SMB\]](#), and thus use it by default.

[<2> Section 2.2.1.1:](#) Windows-based clients always set the **SessionId** field to 0 for these commands, and Windows-based servers will ignore **SessionId** for these commands.

[<3> Section 2.2.2:](#) Windows-based SMB2 servers leave this one byte of **ErrorData** uninitialized and it might contain any value.

[<4> Section 2.2.2.1:](#) Windows servers will never follow a symlink. It is the client's responsibility to evaluate the symlink and access the actual file using the symlink. A Windows server only returns STATUS_STOPPED_ON_SYMLINK when the open fails due to presence of a symlink.

[<5> Section 2.2.2.1:](#) Windows Vista, Windows Server 2008, Windows 7, and Windows Server 2008 R2 will return 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.

[<6> Section 2.2.3:](#) Windows-based SMB2 servers fail the request and return STATUS_INVALID_PARAMETER, if the **DialectCount** field is greater than 64.

[<7> Section 2.2.3:](#) A Windows Vista RTM-based client would send a value of zero in the **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.

[<8> Section 2.2.3:](#) Windows Vista, Windows Server 2008, Windows 7, and Windows Server 2008 R2 support this dialect revision.

- <9> [Section 2.2.3](#): Only Windows 7 and Windows Server 2008 R2 support this dialect revision.
- <10> [Section 2.2.4](#): A Windows Vista RTM-based client would send a value of zero in the **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.
- <11> [Section 2.2.4](#): Windows Vista, Windows Server 2008, Windows 7, and Windows Server 2008 R2 support this dialect revision.
- <12> [Section 2.2.4](#): Only Windows 7 and Windows Server 2008 R2 support this dialect revision.
- <13> [Section 2.2.4](#): The "SMB 2.???" dialect string is supported by SMB2 clients and servers in Windows 7 and Windows Server 2008 R2.
- <14> [Section 2.2.4](#): Windows-based SMB2 servers may set this field to any value.
- <15> [Section 2.2.4](#): Windows-based SMB2 servers generate a new **ServerGuid** each time they are started.
- <16> [Section 2.2.4](#): Windows clients do not enforce the **MaxTransactSize** value.
- <17> [Section 2.2.5](#): Windows-based clients always set the **Capabilities** field to SMB2_GLOBAL_CAP_DFS(0x00000001) and the server will ignore them on receipt.
- <18> [Section 2.2.5](#): Windows clients set the **Buffer** with a token as produced by the NTLM authentication protocol in the case, see [\[MS-NLMP\]](#) section 3.1.5.1.
- <19> [Section 2.2.6](#): Windows clients set the **Buffer** with a token as produced by the NTLM authentication protocol in the case, see [\[MS-NLMP\]](#) section 3.1.5.1.
- <20> [Section 2.2.9](#): The Windows SMB 2 Protocol client translates any names of the form \\server\pipe to \\server\IPC\$ before sending a request on the network.
- <21> [Section 2.2.10](#): SHI1005_FLAGS_FORCE_LEVELII_OPLOCK is not supported on Windows Vista and Windows Server 2008.
- <22> [Section 2.2.13](#): Windows-based clients never use exclusive oplocks. Because there are no situations where the client would require an exclusive oplock where it would not also require an SMB2_OPLOCK_LEVEL_BATCH, it always requests an SMB2_OPLOCK_LEVEL_BATCH.
- <23> [Section 2.2.13](#): When opening an existing file, Windows clients set this field to any value, and Windows-based servers ignore this value.
- <24> [Section 2.2.13](#): When opening a printer file or a named pipe, Windows-based servers ignore these **ShareAccess** values.
- <25> [Section 2.2.13](#): When opening a printer object, Windows-based servers ignore this value.
- <26> [Section 2.2.13](#): When opening a printer object, Windows-based servers ignore this value.
- <27> [Section 2.2.13](#): When opening a printer object, Windows-based servers ignore this value.
- <28> [Section 2.2.13](#): Windows server implementations reserve all bits that are not specified in the table. If any of the reserved bits are set, STATUS_NOT_SUPPORTED is returned.
- <29> [Section 2.2.13](#): Windows SMB2 clients do not initialize this bit. The bit contains the value specified by the caller when requesting the open.

<30> [Section 2.2.13](#): Windows SMB2 clients do not initialize this bit. The bit contains the value specified by the caller when requesting the open.

<31> [Section 2.2.13](#): Windows SMB2 clients do not initialize this bit. The bit contains the value specified by the caller when requesting the open.

<32> [Section 2.2.13](#): Windows SMB2 clients do not initialize this bit. The bit contains the value specified by the caller when requesting the open.

<33> [Section 2.2.13](#): Windows SMB2 clients do not initialize this bit. The bit contains the value specified by the caller when requesting the open.

<34> [Section 2.2.13](#): Windows Vista, Windows Server 2008, and Windows 7-based clients will set this bit when it is requested by the application.

<35> [Section 2.2.13.1.1](#): Windows sets this flag to the value passed in by the higher-level application.

<36> [Section 2.2.13.1.1](#): Windows Server 2008 R2 and Windows 7 do not ignore the SYNCHRONIZE bit, and pass it to the underlying object store. If the caller requests SYNCHRONIZE in the *DesiredAccess* parameter, but the SYNCHRONIZE access is not granted to the caller for the object being created or opened, the underlying object store fails the request and returns STATUS_ACCESS_DENIED. When SYNCHRONIZE access is granted, the SYNCHRONIZE bit is returned in **MaximalAccess** field of SMB2_CREATE_QUERY_MAXIMAL_ACCESS_RESPONSE with no other behavior.

<37> [Section 2.2.13.1.1](#): Windows fails the create request with STATUS_ACCESS_DENIED if the caller does not have the SeSecurityPrivilege, as specified in [\[MS-LSAD\]](#) section 3.1.1.2.1.

<38> [Section 2.2.13.1.2](#): Windows sets this flag to the value passed in by the higher-level application.

<39> [Section 2.2.13.1.2](#): Windows Server 2008 R2 and Windows 7 do not ignore the SYNCHRONIZE bit, and pass it to the underlying object store. If the caller requests SYNCHRONIZE in the **DesiredAccess** parameter, but the SYNCHRONIZE access is not granted to the caller for the object being created or opened, the underlying object store fails the request and returns STATUS_ACCESS_DENIED. When SYNCHRONIZE access is granted, the SYNCHRONIZE bit is returned in **MaximalAccess** field of [SMB2_CREATE_QUERY_MAXIMAL_ACCESS_RESPONSE \(section 2.2.14.2.5\)](#) with no other behavior.

<40> [Section 2.2.13.1.2](#): Windows fails the create request with STATUS_ACCESS_DENIED if the caller does not have the SeSecurityPrivilege, as specified in [\[MS-LSAD\]](#) section 3.1.1.2.1.

<41> [Section 2.2.13.2](#): If DataLength is 0, Windows-based clients set this field to any value.

<42> [Section 2.2.13.2.8](#): Windows 7 and Windows Server 2008 R2 as SMB server support the following combinations of values: 0, READ, READ | WRITE, READ | HANDLE, READ | WRITE | HANDLE. Windows 7 and Windows Server 2008 R2 clients restrict requests accordingly.

<43> [Section 2.2.14](#): Windows-based clients never use exclusive oplocks. Because there are no situations where it would require an exclusive oplock where it would not also require an SMB2_OPLOCK_LEVEL_BATCH, it always requests an SMB2_OPLOCK_LEVEL_BATCH.

<44> [Section 2.2.14](#): Windows-based SMB2 servers always return FILE_OPENED for pipes with successful opens.

<45> [Section 2.2.14](#): Windows-based SMB2 servers may set this field to any value.

<46> [Section 2.2.23.1](#): Windows-based clients never use exclusive oplocks. Because there are no situations where it would require an exclusive oplock where it would not also require an SMB2_OPLOCK_LEVEL_BATCH, it always requests an SMB2_OPLOCK_LEVEL_BATCH.

<47> [Section 2.2.24.1](#): 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.

<48> [Section 2.2.24.2](#): Windows clients always set the LeaseState in the Lease Break Acknowledgment to be equal to the LeaseState in the [Lease Break Notification](#) from the server.

<49> [Section 2.2.31](#): If no input data is required for the FSCTL/IOCTL command being issued, Windows-based clients set this field to any value.

<50> [Section 2.2.31](#): Windows clients set the **OutputOffset** field equal to the **InputOffset** field.

<51> [Section 2.2.31.1.1](#): Windows sets this field to any value.

<52> [Section 2.2.31.2.1](#): Windows-based SMB2 servers and clients do not check **SourceFileName**. It is ignored.

<53> [Section 2.2.32](#): Windows-based SMB2 servers set **InputCount** to the same value as the value received in the IOCTL request for the following FSCTLs.

- FSCTL_FIND_FILES_BY_SID
- FSCTL_GET_RETRIEVAL_POINTERS
- FSCTL_QUERY_ALLOCATED_RANGES
- FSCTL_READ_FILE_USN_DATA
- FSCTL_RECALL_FILE
- FSCTL_WRITE_USN_CLOSE_RECORD

Windows clients ignore the **InputCount** field.

<54> [Section 2.2.32](#): Windows-based SMB2 servers set **OutputOffset** to **InputOffset** + **InputCount**, rounded up to a multiple of 8.

<55> [Section 2.2.32.2](#): Windows-based SMB2 server will place 2 extra bytes set to zero in the SRV_SNAPSHOT_ARRAY response, if **NumberOfSnapshotsReturned** is zero.

<56> [Section 2.2.32.3](#): Windows-based servers Windows Vista, Windows Server 2008, Windows 7, and Windows Server 2008 R2 always send 4 bytes of zero for the Context field.

<57> [Section 2.2.33](#): Windows-based servers do not support resuming an enumeration at a specified **FileIndex**. The server will ignore this flag.

<58> [Section 2.2.33](#): SMB2 wildcard characters are based on Windows wildcard characters, as described in [\[MS-FSA\]](#) section 3.1.4.4, Algorithm for Determining if a FileName Is in an Expression. For more information on wildcard behavior in Windows, see [\[FSBO\]](#) section 7.

<59> [Section 2.2.37](#): Windows SMB2 servers ignore the **FileInfoClass** field for quota queries. Windows SMB2 clients set the **FileInfoClass** field to 0x20 for quota queries.

<60> [Section 2.2.37](#): Windows clients set this value to the offset from the start of the [SMB2 header](#) to the beginning of the **Buffer** field.

<61> [Section 2.2.37:](#) Windows clients send a 1-byte buffer of 0 when **InputBufferLength** is set to 0.

<62> [Section 2.2.37.1:](#) Windows clients set this field to 1 for TRUE.

<63> [Section 2.2.37.1:](#) Windows clients set this field to 1 for TRUE.

<64> [Section 2.2.37.1:](#) Windows Vista, Windows Server 2008, Windows 7, and Windows Server 2008 R2 clients never send a request using the **SidBuffer** format 2.

<65> [Section 3.1.3:](#) By default, Windows-based servers set the [RequireMessageSigning](#) value to TRUE for domain controllers and FALSE for all other machines.

<66> [Section 3.2.1.2:](#) Windows clients do not enforce the **MaxTransactSize** value.

<67> [Section 3.2.2.1:](#) The Windows-based client implements this timer with a default value of 60 seconds. The client does not enforce this timer for the following commands:

- Named Pipe Read
- Named Pipe Write
- Directory Change Notifications
- Blocking byte range lock requests
- FSCTLs: FSCTL_PIPE_PEEK, FSCTL_PIPE_TRANSCEIVE, FSCTL_PIPE_WAIT

<68> [Section 3.2.2.2:](#) The Windows-based clients scan existing connections every 10 seconds and disconnect idle connects that have no open files and that have had no activity for 10 or more seconds.

<69> [Section 3.2.4.1.1:](#) A client can selectively sign requests, and the server will sign the corresponding responses. Windows-based clients do not selectively sign requests.

<70> [Section 3.2.4.1.2:](#) Windows Vista SP1, Windows 7, Windows Server 2008, and Windows Server 2008 R2 clients require a minimum of 4 credits.

<71> [Section 3.2.4.1.2:](#) The Windows-based client will request credits up to a configurable maximum of 128 by default. A Windows-based client sends a **CreditRequest** value of 0 for an [SMB2 NEGOTIATE Request](#) and expects the server to grant at least 1 credit. In subsequent requests, the client will request credits sufficient to maintain its total outstanding limit at the configured maximum.

<72> [Section 3.2.4.1.3:](#) Windows 7 and Windows Server 2008 R2 SMB2 clients will block any newly initiated multi-credit requests which exceed the shortage, but will send out other requests which can be satisfied using the available credits.

<73> [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 the start of a chain.

<74> [Section 3.2.4.1.4:](#) Windows Vista, Windows Server 2008, Windows 7, and Windows Server 2008 R2 will align their compounded requests and responses on 8-byte boundaries. Currently they do not disconnect other machines that disobey this rule.

<75> [Section 3.2.4.1.4:](#) The Windows-based client does not send unrelated compounded requests.

[<76> Section 3.2.4.1.4:](#) Windows-based clients will compound certain related requests to improve performance, by combining a Create with another operation, such as an attribute query.

[<77> Section 3.2.4.1.5:](#) Windows 7 and Windows Server 2008 R2 SMB2 clients send multi-credit requests only for read, write, and directory enumeration. For all other requests, a Windows-based client uses credit charge 1, even if the payload size of a request or the anticipated response is greater than 64 kilobytes.

[<78> Section 3.2.4.1.5:](#) Windows 7-based SMB2 clients set the **CreditCharge** field to 1 if **Connection.SupportsMultiCredit** is FALSE.

[<79> Section 3.2.4.2:](#) Windows-based clients always set up a new transport connection when establishing a new session to a server.

[<80> Section 3.2.4.2:](#) Windows will reuse an existing session if the access is by the same logged-on user and the target server name matches exactly. This means that Windows will establish a new session with the same credentials if the same user is logged on to the client multiple times, or if the user is accessing the server through two different names that resolve to the same server. (NetBIOS and fully qualified domain name, for example.)

[<81> Section 3.2.4.2:](#) Windows will establish a new connection for every SMB2 session being created.

[<82> Section 3.2.4.2:](#) Windows establishes a new connection for each new session.

[<83> Section 3.2.4.2.1:](#) Windows 7 and Windows Server 2008 R2 attempt to connect to the first eligible address returned from name resolution, preferring all transports supporting Direct TCP, and after 1000 milliseconds will attempt to connect to all other eligible addresses and transports in parallel. Windows Vista and Windows Server 2008 attempt to connect to the first eligible address, preferring a single Direct TCP transport, and after 500 milliseconds will attempt to connect to all other eligible addresses and all other NetBIOS over TCP transports. In each case, the first successful connection is used and all others are closed.

[<84> 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 Protocol. In the latter case, it will initiate an [SMB2-only negotiate](#).

[<85> Section 3.2.4.2.2.1:](#) When a Windows-based client sends the deprecated "SMB 2.001" dialect, a Windows Vista RTM-based server would acknowledge with a value of 6 in *DialectRevision* in the [SMB2 NEGOTIATE Response](#). This behavior is deprecated.

[<86> Section 3.2.4.2.3:](#) Windows-based clients implement the first option that is specified.

[<87> Section 3.2.4.2.3:](#) All the GSS-API tokens used by Windows SMB2 clients are up to 4Kbytes in size. SMB2 servers always instruct the GSS_API server to expect the *GSS_C_FRAGMENT_TO_FIT*.

[<88> Section 3.2.4.2.3.1:](#) Windows-based clients implement the first option that is specified.

[<89> Section 3.2.4.2.3.1:](#) All the GSS-API tokens used by Windows SMB2 clients are up to 4Kbytes in size. SMB2 servers always instruct the GSS_API server to expect the *GSS_C_FRAGMENT_TO_FIT*.

[<90> Section 3.2.4.3:](#) Windows-based clients will request a batch oplock for file creates.

[<91> Section 3.2.4.3.8:](#) A Windows client application requests *SMB2_LEASE_READ_CACHING* and *SMB2_LEASE_HANDLE_CACHING* when a file is opened for read access. In addition, a Windows client application requests *SMB2_LEASE_WRITE_CACHING* if the file is being opened for write access.

<92> [Section 3.2.4.6](#): Windows-based clients will try to send multiple read commands at the same time, starting at the lowest offset and working to the highest.

<93> [Section 3.2.4.7](#): Windows-based clients always put the payload at the beginning of the **Buffer** field and do not insert padding.

<94> [Section 3.2.4.7](#): Windows-based clients will try to send multiple write commands at the same time, starting at the lowest offset and working to the highest.

<95> [Section 3.2.4.8](#): Windows clients set this value to the offset from the start of the [SMB2 header](#) to the beginning of the **Buffer** field.

<96> [Section 3.2.4.10](#): Windows clients set this value to the offset from the start of the [SMB2 header](#) to the beginning of the **Buffer** field.

<97> [Section 3.2.4.12](#): Windows clients set this value to the offset from the start of the [SMB2 header](#) to the beginning of the **Buffer** field.

<98> [Section 3.2.4.14](#): Windows Vista, Windows Server 2008, Windows 7, and Windows Server 2008 R2 clients will set **StartSidLength** and **StartSidOffset** to any value.

<99> [Section 3.2.4.17](#): The Windows SMB2 server implementation closes and reopens the directory handle in order to "reset" the enumeration state. So any outstanding operations on the directory handle will be failed with a STATUS_FILE_CLOSED error.

<100> [Section 3.2.4.20.1](#): Windows clients set this field to any value.

<101> [Section 3.2.4.20.1](#): Windows clients set the **OutputOffset** field to **InputOffset** + **InputCount**, rounded up to a multiple of 8 bytes.

<102> [Section 3.2.4.20.2.1](#): Windows clients set this field to any value.

<103> [Section 3.2.4.20.2.1](#): Windows clients set this field to **InputOffset** + **InputCount**, rounded up to a multiple of 8 bytes.

<104> [Section 3.2.4.20.2.2](#): Windows applications use FSCTL_SRV_COPYCHUNK if the target file handle has FILE_READ_DATA access. Otherwise, they use the FSCTL_SRV_COPYCHUNK_WRITE.

<105> [Section 3.2.4.20.2.2](#): Windows clients set the **OutputOffset** field to **InputOffset** + **InputCount**, rounded up to a multiple of 8 bytes.

<106> [Section 3.2.4.20.3](#): Windows clients set the **OutputOffset** field to **InputOffset** + **InputCount**, rounded up to a multiple of 8 bytes.

<107> [Section 3.2.4.20.4](#): Windows clients set the **OutputOffset** field to **InputOffset** + **InputCount**, rounded up to a multiple of 8 bytes.

<108> [Section 3.2.4.20.5](#): Windows clients set this field to any value.

<109> [Section 3.2.4.20.5](#): Windows clients set the **OutputOffset** field to **InputOffset** + **InputCount**, rounded up to a multiple of 8 bytes.

<110> [Section 3.2.4.20.6](#): Windows-based SMB2 servers pass File System Control requests through to the **local object store** but do not support I/O Control requests and fail such requests with STATUS_NOT_SUPPORTED.

<111> [Section 3.2.4.20.6](#): Windows clients set the **OutputOffset** field to **InputOffset** + **InputCount**, rounded up to a multiple of 8 bytes.

<112> [Section 3.2.4.20.7](#): Windows clients set the **OutputOffset** field to the sum of the values of the **InputOffset** and the **InputCount** fields, rounded up to a multiple of 8 bytes.

<113> [Section 3.2.4.20.8](#): Windows clients set the **OutputOffset** field to **InputOffset** + **InputCount**, rounded up to a multiple of 8 bytes.

<114> [Section 3.2.4.24](#): Windows based clients set the **MessageId** field to 0, when the **AsyncId** field is set to an asynchronous identifier of the request.

<115> [Section 3.2.4.24](#): Windows-based clients set the **SessionId** field to **TreeConnect.Session.SessionId**, and Windows-based servers will ignore the **SessionId** field.

<116> [Section 3.2.5.1.2](#): Windows-based clients will not disconnect the connection but simply disregard the incorrectly signed response.

<117> [Section 3.2.5.1.6](#): Windows-based clients will not disconnect the connection, but will simply fail the request.

<118> [Section 3.2.5.1.7](#): Windows-based SMB 2 Protocol clients do not check the validity of the command in the response.

<119> [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.

<120> [Section 3.2.5.14](#): If the **OutputCount** field in an [SMB2_IOCTL Response](#) is 0 and the **OutputOffset** exceeds the size of the SMB2 response, Windows clients will return STATUS_INVALID_NETWORK_RESPONSE to the application.

<121> [Section 3.2.5.14.9](#): Windows clients enable TCP keepalives to detect broken connections.

<122> [Section 3.2.5.19.1](#): Windows-based clients will not request exclusive oplocks.

<123> [Section 3.2.6.1](#): Windows clients use a default time-out of 60 seconds for all requests with the following exception. Windows clients do not enforce a time-out on SMB2 CHANGE_NOTIFY requests, SMB2 LOCK requests without the SMB2_LOCKFLAG_FAIL_IMMEDIATELY flag, SMB2 READ requests on named pipes, SMB2 WRITE requests on named pipes, and the FSCTL_PIPE_PEEK, FSCTL_PIPE_TRANSCEIVE and FSCTL_PIPE_WAIT named pipe FSCTLs. Windows clients enforce a time-out on SMB2 FLUSH requests on named pipes.

<124> [Section 3.2.6.1](#): Windows clients do not extend the default time-out.

<125> [Section 3.2.6.1](#): Windows-based clients return a STATUS_CONNECTION_DISCONNECTED error code to the calling application.

<126> [Section 3.2.6.1](#): The Windows-based clients will disconnect the connection.

<127> [Section 3.3.1.1](#): 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 $2 \times 10 = 20$. 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.

<128> [Section 3.3.1.2](#): 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.

[<129> Section 3.3.1.2:](#) Windows Vista SP1, Windows 7, Windows Server 2008, and Windows Server 2008 R2 SMB2 servers support a configurable minimum credit limit below which the client is unconditionally granted all credits it requests, and a configurable maximum credit limit above which credits are never granted, as follows:

SMB2 server	Default minimum	Default maximum
Windows Vista SP1 and Windows 7	128	2048
Windows Server 2008 and Windows Server 2008 R2	512	8192

[<130> Section 3.3.1.2:](#) A Windows-based server does not currently scale credits based on quality of service features.

[<131> Section 3.3.1.4:](#) Windows 7 and Windows Server 2008 R2-based SMB2 servers support only the levels described above, and Windows 7 and Windows Server 2008 R2-based SMB2 clients request only those levels.

[<132> Section 3.3.1.6:](#) Windows-based servers allow the sharing of both printers and traditional file shares.

[<133> Section 3.3.1.6:](#) In Windows, this abstract state element contains the security descriptor for the share.

[<134> Section 3.3.1.6:](#) Windows-based SMB2 clients do not cache directory enumeration results.

[<135> Section 3.3.1.10:](#) Windows 7 and Windows Server 2008 R2 SMB 2.1 server support a maximum extent of 64 entries in the **Open.LockSequenceArray[]**.

[<136> Section 3.3.1.13:](#) The Windows SMB2 server allocates an I/O request (IRP) structure which it uses to locally request action from the object store. The **Request.CancelRequestId** is set to the unique address of this structure.

[<137> 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).

[<138> Section 3.3.2.2:](#) Windows-based SMB2 servers set this timer to a constant value of 16 minutes.

[<139> Section 3.3.2.3:](#) Windows-based servers implement this timer with a constant value of 45 seconds.

[<140> Section 3.3.3:](#) Windows-based SMB2 servers set this value to 256.

[<141> Section 3.3.3:](#) Windows-based SMB2 servers set this value to 1 MB.

[<142> Section 3.3.3:](#) Windows-based SMB2 servers set this value to 16 MB.

[<143> Section 3.3.3:](#) Windows servers initialize **ServerHashLevel** based on a stored value in the registry.

[<144> Section 3.3.3:](#) Windows 7 and Windows Server 2008 R2 SMB2 servers provide a constant maximum resiliency time-out of 300000 milliseconds.

[<145> Section 3.3.4.1.1:](#) Windows-based servers do not sign interim responses.

[<146> Section 3.3.4.1.2:](#) For an asynchronously processed request, Windows Vista, Windows Server 2008, Windows 7, and Windows Server 2008 R2 grant credits on the interim response and do not grant credits on the final response. The interim response grants credits to keep the transaction from stalling in case the client is out of credits.

[<147> Section 3.3.4.1.3:](#) The Windows-based server compounds responses for any received compounded operations. Otherwise, it does not compound responses.

[<148> Section 3.3.4.1.3:](#) Windows-based servers grant all credits in the final response of the compounded chain, and grant 0 credits in all responses other than the final response.

[<149> 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
- Large file write
- FSCTL_PIPE_TRANSCEIVE
- FSCTL_SRV_COPYCHUNK or FSCTL_SRV_COPYCHUNK_WRITE, when oplock break happens
- [SMB2_FLUSH](#) on a named pipe

[<150> Section 3.3.4.2:](#) Windows-based servers incorrectly process the FSCTL_PIPE_WAIT request on named pipes synchronously.

[<151> Section 3.3.4.2:](#) Windows servers enforce a configurable blocking operation credit, which defaults to 64 on Windows Vista SP1 and Windows 7, and defaults to 512 on Windows Server 2008 and Windows Server 2008 R2.

[<152> Section 3.3.4.2:](#) Windows servers always fail requests that go to async mode in a compound request with STATUS_INTERNAL_ERROR 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 requires to go async mode. If a create request in a compound chain goes to async mode due to oplock break, Windows servers only send one interim response to the client. If there are other create requests in a compound chain already completed before the oplock break, the broken oplock level will not be updated in these create responses in the compound chain.

[<153> Section 3.3.4.4:](#) For Windows 7 and Windows Server 2008 R2, STATUS_BUFFER_OVERFLOW will be returned for FSCTL_GET_RETRIEVAL_POINTERS and FSCTL_GET_REPARSE_POINT, along with the ones mentioned in section [3.3.4.4](#).

[<154> Section 3.3.4.6:](#) Windows based SMB2 servers set **Open.OplockTimeout** to the current time plus 35000 milliseconds.

[<155> Section 3.3.4.7:](#) Windows based SMB2 servers set **Lease.LeaseBreakTimeout** to the current time plus 35000 milliseconds.

<156> [Section 3.3.4.17](#): Windows Lease break is described in [\[MS-FSA\]](#) section 3.1.5.17. The *Open* parameter passed is the **Open** value from the current close operation, the *Type* parameter is LEVEL_GRANULAR to indicate a Lease request, and the *RequestedOplockLevel* parameter is zero.

<157> [Section 3.3.4.21](#): For each supported transport type as listed in section [2.1](#), the Windows SMB2 server attempts to form an association with the specified device with local calls specific to each supported transport type and rejects the entry if none of the associations succeed.

<158> [Section 3.3.4.21](#): On Windows, **ServerName** is used only when the transport is NBT.

<159> [Section 3.3.5.1](#): Possible Windows-specific values for **Connection.TransportName** are listed in a product behavior note attached to [\[MS-SRVS\]](#) section 2.2.4.96.

<160> [Section 3.3.5.2](#): Windows performs cancellation of in-progress requests via the interface in [\[MS-FSA\]](#) section 3.1.5.19, Server Requests Canceling an Operation, passing **Request.CancelRequestId** as an input parameter.

<161> [Section 3.3.5.2.2](#): Windows-based servers will not fail the request with an error code but will disconnect clients that send noncompounded requests with an invalid **MessageId**. When the **MessageId** is invalid in the compounded requests, the server behavior is undefined.

<162> [Section 3.3.5.2.3](#): Windows-based servers will not disconnect the connection due to a mismatched signature.

<163> [Section 3.3.5.2.3](#): Windows-based servers will not disconnect the connection due to an unsigned packet.

<164> [Section 3.3.5.2.4](#): Windows Vista and Windows Server 2008 do not verify the payload size of the request.

<165> [Section 3.3.5.2.5](#): 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.

<166> [Section 3.3.5.2.6](#): Windows-based SMB2 servers always fail requests that go to async mode in a compound request with STATUS_INTERNAL_ERROR 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 goes to async mode. If multiple requests in a compound message go to async mode, only the first request will be responded to by Windows-based SMB2 servers. Subsequent async requests will be ignored by Windows-based SMB2 server implementations. Responses for the requests prior to the first async request are sent as part of a single response. Other responses are returned in separate messages.

<167> [Section 3.3.5.2.6](#): Windows-based SMB2 servers allow a mix of related and unrelated compound requests in the same transport send. Upon encountering a request with SMB2_FLAGS_RELATED_OPERATIONS not set, a Windows-based SMB2 server treats it as the start of a chain.

<168> [Section 3.3.5.2.6.2](#): If SMB2_FLAGS_RELATED_OPERATIONS is present in the first request and the request does not have valid **SessionId**, **TreeId** or **FileId**, Windows servers fail all requests in compounded chain with error STATUS_INVALID_PARAMETER. Otherwise, the operation will succeed.

<169> [Section 3.3.5.2.6.2](#): Windows Vista, Windows Server 2008, Windows 7, and Windows Server 2008 R2 fail the compounded request with STATUS_INVALID_PARAMETER if the previous request failed to create the **FileId** or the compounded request does not contain a **FileId**, **SessionId**, or **TreeId**. If the previous session expired, Windows Vista, Windows Server 2008,

Windows 7, and Windows Server 2008 R2 fail the next request in the compound request with STATUS_NETWORK_SESSION_EXPIRED, and the subsequent requests in the compounded request will be failed with STATUS_INVALID_PARAMETER.

<170> [Section 3.3.5.3.1](#): If **Connection.SupportsMultiCredit** is TRUE, this is set to a default value of 1024KB. Otherwise, this is set to a default value of 64KB.

<171> [Section 3.3.5.3.1](#): If the negotiate dialect is SMB 2.1 and the underlying TCP endpoint is on port 445, the default **MaxReadSize** value is set to 1024KB. This default value can be customized through registry configurations. Otherwise, the default **MaxReadSize** value is set to 64KB.

<172> [Section 3.3.5.3.1](#): If the negotiate dialect is SMB 2.1 and the underlying TCP endpoint is on port 445, the default **MaxWriteSize** value is set to 1024KB. This default value can be customized through registry configurations. Otherwise, the default **MaxWriteSize** value is set to 64K.

<173> [Section 3.3.5.3.2](#): When a Windows-based client sends the deprecated "SMB 2.001" dialect, a Windows Vista RTM-based server would acknowledge with a value of 6 in **DialectRevision** in the [SMB2 NEGOTIATE Response](#). This behavior is deprecated.

<174> [Section 3.3.5.3.2](#): A Windows Vista RTM-based server sets **DialectRevision** to 6.

<175> [Section 3.3.5.3.2](#): Windows servers set this to a default value of 64KB.

<176> [Section 3.3.5.4](#): If **Connection.SupportsMultiCredit** is TRUE, this is set to a default value of 1024KB. Otherwise, this is set to a default value of 64KB.

<177> [Section 3.3.5.5.3](#): Windows 2000, Windows XP, Windows Server 2003, Windows Vista, Windows Server 2008, Windows 7, and Windows Server 2008 R2 will also accept raw Kerberos messages and implicit NTLM messages as part of GSS authentication.

<178> [Section 3.3.5.5.3](#): Windows by default uses the **guest account** to represent guest users. Alternatively, any user account that is a member of the well-known BUILTIN_GUESTS or DOMAIN_GUESTS group (see [\[MS-DTYP\]](#) section 2.4.2.4) is considered a guest account.

<179> [Section 3.3.5.7](#): Windows-based SMB2 servers do not set this bit in the **ShareFlags** field.

<180> [Section 3.3.5.7](#): Windows-based SMB2 servers do not set this bit in the **ShareFlags** field.

<181> [Section 3.3.5.9](#): If the target of the create request is a file, and any of the bits in the mask 0x0CE0FE00 are set, Windows allows the file open but does not allow any operations on the opened file.

<182> [Section 3.3.5.9](#): Windows-based servers ignore **DesiredAccess** values other than FILE_WRITE_DATA, FILE_APPEND_DATA and GENERIC_WRITE if any one of these values is specified.

<183> [Section 3.3.5.9](#): Windows-based servers fail requests having a **CreatedDisposition** of FILE_OPEN or FILE_OVERWRITE, but ignore values of FILE_SUPERSEDE, FILE_OPEN_IF and FILE_OVERWRITE_IF.

<184> [Section 3.3.5.9](#): Windows Vista, Windows Server 2008, Windows 7, and Windows Server 2008 R2 ignore create contexts having a **NameLength** greater than 4 and ignores create contexts with length of 4 that are not specified in section [2.2.13.2](#).

<185> [Section 3.3.5.9](#): Windows-based SMB2 servers check only for FILE_WRITE_DATA, FILE_WRITE_ATTRIBUTES, FILE_WRITE_EA, and FILE_APPEND_DATA in the **DesiredAccess** field.

<186> [Section 3.3.5.9](#): Windows Vista and Windows Server 2008 do not support the SHI1005_FLAGS_FORCE_LEVELII_OPLOCK flag and ignore the **TreeConnect.Share.ForceLevel2Oplock** value.

<187> [Section 3.3.5.9](#): Windows performs the following open/create mappings from SMB2 parameters to the object store as described in [\[MS-FSA\]](#) section 3.1.5.1 Server Requests an Open of a File.

Object Store parameter	SMB2 parameter	Notes
DesiredAccess	DesiredAccess	
DesiredFileAttributes	FileAttributes	
ShareAccess	ShareAccess	
CreateDisposition	CreateDisposition	
CreateOptions	CreateOptions	
SecurityContext	Session.SecurityContext SecurityFlags ImpersonationLevel	SecurityFlags and ImpersonationLevel are not passed to the object store
PathName	PathName	Relative to TreeConnect.Share.LocalPath
RootOpen	TreeConnect.Share	Share open instance obtained at SMB2 initialization
IsCaseSensitive	FALSE	Windows-based SMB2 servers always handle path names as case-insensitive
OplockKey	NULL	OplockKey specified only for obtaining Leases

Windows performs the following mappings from object store results to SMB2 response.

Object Store result	SMB2 response	Notes
CreateAction	CreateAction	
Open	FileId	The FileId to Open mapping is computed and maintained by the server

<188> [Section 3.3.5.9](#): Windows-based servers will receive the data from the local create operation for constructing the error response when a symbolic link is present in the target path name.

<189> [Section 3.3.5.9](#): Windows **Oplock** acquisition is described in [\[MS-FSA\]](#) section 3.1.5.17. Oplock acquisition is an optional step in open/create processing; the Open parameter passed is the Open result from the open or create operation, and the Type parameter is mapped as follows.

Object Store oplock Type	SMB2 oplock level
LEVEL_BATCH	SMB2_OPLOCK_LEVEL_BATCH
LEVEL_ONE	SMB2_OPLOCK_LEVEL_EXCLUSIVE

Object Store oplock Type	SMB2 oplock level
LEVEL_TWO	SMB2_OPLOCK_LEVEL_II

The **Status** code returned indicates whether the requested oplock was granted.

<190> [Section 3.3.5.9](#): Windows obtains **CreationTime** from the object store FileBasicInformation [\[MS-FSA\]](#) section 3.1.5.11.6 and [\[MS-FSCC\]](#) section 2.4.7.

<191> [Section 3.3.5.9](#): Windows obtains **LastAccessTime** from the object store FileBasicInformation [\[MS-FSA\]](#) section 3.1.5.11.6 and [\[MS-FSCC\]](#) section 2.4.7.

<192> [Section 3.3.5.9](#): Windows obtains **LastWriteTime** from the object store FileBasicInformation [\[MS-FSA\]](#) section 3.1.5.11.6 and [\[MS-FSCC\]](#) section 2.4.7.

<193> [Section 3.3.5.9](#): Windows obtains **ChangeTime** from the object store FileBasicInformation [\[MS-FSA\]](#) section 3.1.5.11.6 and [\[MS-FSCC\]](#) section 2.4.7.

<194> [Section 3.3.5.9](#): Windows obtains **AllocationSize** from the object store FileStandardInformation [\[MS-FSA\]](#) section 3.1.5.11.27 and [\[MS-FSCC\]](#) section 2.4.38.

<195> [Section 3.3.5.9](#): Windows-based SMB2 servers will set AllocationSize to any value for the named pipe.

<196> [Section 3.3.5.9](#): Windows obtains **EndOfFile** from the object store FileStandardInformation [\[MS-FSA\]](#) section 3.1.5.11.27 and [\[MS-FSCC\]](#) section 2.4.38.

<197> [Section 3.3.5.9](#): Windows-based SMB2 servers will set EndOfFile to any value for the named pipe.

<198> [Section 3.3.5.9](#): Windows obtains **FileAttributes** from the object store FileBasicInformation [\[MS-FSA\]](#) section 3.1.5.11.6 and [\[MS-FSCC\]](#) section 2.4.7.

<199> [Section 3.3.5.9](#): Windows servers will process and respond to an SMB2_CREATE_QUERY_MAXIMAL_ACCESS_REQUEST context regardless of whether an SMB2_CREATE_DURABLE_HANDLE_RECONNECT context is present.

<200> [Section 3.3.5.9.1](#): Windows sets extended attributes on a newly created file with the **FSCTL_SET_OBJECT_ID_EXTENDED** [FSCTL](#) [\[MS-FSA\]](#) section 3.1.5.9.24 and [\[MS-FSCC\]](#) section 2.3.53.

<201> [Section 3.3.5.9.2](#): Windows sets security attributes on a newly created file with the Application requests setting of security information [\[MS-FSA\]](#) section 3.1.5.16.

<202> [Section 3.3.5.9.2](#): Windows will ignore security descriptors if the underlying object store does not support them.

<203> [Section 3.3.5.9.3](#): Windows-based servers support this request.

<204> [Section 3.3.5.9.3](#): Windows sets allocation size on a newly created file with the FileAllocationInformation [\[MS-FSA\]](#) section 3.1.5.14.1 and [\[MS-FSCC\]](#) section 2.4.4, after converting bytes to volume cluster size.

<205> [Section 3.3.5.9.4](#): Windows validates that a snapshot with the time stamp provided exists by forming a **FileBothDirectoryInformation** object store request for the file including the provided @GMT token in the path, as described in [\[MS-SMB\]](#) section 2.2.1.1.1 and [\[MS-FSA\]](#) section 3.1.5.4.1.

<206> [Section 3.3.5.9.4](#): Windows opens a file on a snapshot with the time stamp provided by the file including the provided @GMT token in the path, as described in [\[MS-SMB\]](#) section 2.2.1.1.1 and [\[MS-FSA\]](#) section 3.1.5.1.

<207> [Section 3.3.5.9.5](#): Windows computes the MaximalAccess to return by querying the security attributes of the file with [\[MS-FSA\]](#) section 3.1.5.13, and performing an access check against the credentials provided by the request.

<208> [Section 3.3.5.9.8](#): The **Lease.LeaseKey** generated in section [3.3.5.9.8](#) is associated with the **LeaseTable.ClientGuid** to generate a unique **OplockKey** which is passed to the object store when processing continues at open/create time. A new or existing lease is thereby associated with the resulting open.

To acquire or promote the lease as dictated by the SMB2_CREATE_REQUEST_LIST Create Context, a subsequent object store call is invoked as described in [\[MS-FSA\]](#) section 3.1.5.17 Server Requests an Oplock. The Open parameter passed is the Open result from the above operation, and the Type parameter is LEVEL_GRANULAR to indicate a Lease request. The **RequestedOplockLevel** parameter is constructed to include zero or more bits as follows.

Object Store RequestedOplockLevel bit to be set	SMB2 Lease.LeaseState bit requested
READ_CACHING	SMB2_LEASE_READ_CACHING
WRITE_CACHING	SMB2_LEASE_WRITE_CACHING
HANDLE_CACHING	SMB2_LEASE_HANDLE_CACHING

The Status code returned indicates whether the requested lease was granted.

<209> [Section 3.3.5.9.9](#): Windows SMB2 servers generate the value using the **Open.FileId** of the file and volatile local information specific to the mounted volume. If the volume is remounted or the server is restarted, the value will change.

<210> [Section 3.3.5.10](#): Windows obtains attributes and end of file from the object store FileBasicInformation [\[MS-FSA\]](#) section 3.1.5.11.6 and [\[MS-FSCC\]](#) section 2.4.7.

<211> [Section 3.3.5.11](#): Windows flushes any cached data to the file with Server Requests Flushing Cached Data [\[MS-FSA\]](#) section 3.1.5.6.

<212> [Section 3.3.5.11](#): If the request target is a named pipe or file, Windows-based servers handle this request asynchronously.

<213> [Section 3.3.5.12](#): Windows Vista, Windows Server 2008, Windows 7, and Windows Server 2008 R2 fail the request with STATUS_BUFFER_OVERFLOW instead of STATUS_INVALID_PARAMETER.

<214> [Section 3.3.5.12](#): Windows reads from a file with Server Requests a Read [\[MS-FSA\]](#) section 3.1.5.2.

Object Store parameter	SMB2 parameter
ByteOffset	ByteOffset
ByteCount	ByteCount
IsNonCached	FALSE

<215> [Section 3.3.5.12](#): Windows SMB2 servers send an interim response to the client and handle the read asynchronously if the read is not finished in 0.5 milliseconds.

<216> [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 IOCTL Request](#) for FSCTL_SRV_COPYCHUNK or FSCTL_SRV_COPYCHUNK_WRITE (section 2.2.31) when oplock break happens, [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 milliseconds, [SMB2 Write Request \(section 2.2.21\)](#) for large file write, [SMB2 lock request \(section 2.2.26\)](#) if the SMB2_LOCKFLAG_FAIL_IMMEDIATELY flag is not set, and [SMB2 FLUSH Request \(section 2.2.17\)](#) for named pipes.

<217> [Section 3.3.5.13](#): Windows SMB2 servers allow the operation when either FILE_APPEND_DATA or FILE_WRITE_DATA is set in **Open.GrantedAccess**.

<218> [Section 3.3.5.13](#): Windows Vista, Windows Server 2008, Windows 7, and Windows Server 2008 R2 fail the request with STATUS_BUFFER_OVERFLOW instead of STATUS_INVALID_PARAMETER.

<219> [Section 3.3.5.13](#): Windows writes to a file with Server Requests a Write [\[MS-FSA\]](#) section 3.1.5.3.

Object Store parameter	SMB2 parameter
ByteOffset	ByteOffset
ByteCount	ByteCount
IsWriteThrough	TRUE if SMB2_WRITEFLAG_WRITE_THROUGH flag set
IsNonCached	FALSE

<220> [Section 3.3.5.13](#): Windows-based servers handle the following commands asynchronously:

- [SMB2 CREATE Request](#) (section [3.3.5.9](#)) when this create would result in an oplock break.
- [SMB2 IOCTL Request](#) (section [3.3.5.15](#)) for FSCTL_PIPE_TRANSCEIVE if it blocks for more than 1 millisecond. For FSCTL_SRV_COPYCHUNK or FSCTL_SRV_COPYCHUNK_WRITE, when an oplock break happens.
- [SMB2 CHANGE NOTIFY Request](#) (section [3.3.5.19](#)) if it blocks for more than 0.5 milliseconds.
- [SMB2 READ Request](#) (section [3.3.5.12](#)) for named pipes if it blocks for more than 0.5 milliseconds.
- [SMB2 WRITE Request](#) (section [3.3.5.13](#)) for named pipes if it blocks for more than 0.5 milliseconds.
- [SMB2 WRITE Request](#) (section [3.3.5.13](#)) for large file write.
- [SMB2 LOCK Request](#) (section [3.3.5.14](#)) if the SMB2_LOCKFLAG_FAIL_IMMEDIATELY flag is not set.
- [SMB2 FLUSH Request](#) (section [3.3.5.11](#)) for named pipes.

<221> [Section 3.3.5.14.1](#): Windows-based servers ignore this value while processing Unlocks.

<222> [Section 3.3.5.14.1](#): Windows processes unlock with Server Requests unlock of a Byte-Range [MS-FSA] section 3.1.5.8.

Object Store parameter	SMB2 parameter
FileOffset	Offset
Length	Length

<223> [Section 3.3.5.14.2](#): Windows-based servers check for SMB2_LOCKFLAG_FAIL_IMMEDIATELY only for the first element of the **Locks** array.

<224> [Section 3.3.5.14.2](#): Refer to [FSBO] for implementation-specific details of how byte range locks can be implemented.

<225> [Section 3.3.5.14.2](#): Windows processes lock with Server Requests a Byte-Range Lock [MS-FSA] section 3.1.5.7.

Object Store parameter	SMB2 parameter
FileOffset	Offset
Length	Length
ExclusiveLock	FALSE if SMB2_LOCKFLAG_SHARED_LOCK set, or TRUE if SMB2_LOCKFLAG_EXCLUSIVE_LOCK set
FailImmediately	TRUE if SMB2_LOCKFLAG_FAIL_IMMEDIATELY set

<226> [Section 3.3.5.15](#): Windows 7 and Windows Server 2008 R2 SMB2 servers copy the **OutputCount** bytes into the output buffer for the following FSCTLs:

- FSCTL_GET_RETRIEVAL_POINTERS
- FSCTL_GET_REPARSE_POINT
- FSCTL_PIPE_TRANSCEIVE
- FSCTL_PIPE_PEEK
- FSCTL_DFS_GET_REFERRALS

Windows Vista and Windows Server 2008 SMB2 servers copy the **OutputCount** bytes into the output buffer for the following FSCTLs:

- FSCTL_PIPE_TRANSCEIVE
- FSCTL_PIPE_PEEK
- FSCTL_DFS_GET_REFERRALS

All other FSCTL commands will be failed with error STATUS_BUFFER_OVERFLOW through error response specified in section [2.2.2](#).

<227> [Section 3.3.5.15](#): Windows Vista and Windows Server 2008 servers without KB 978491, and Windows 7 and Windows Server 2008 R2 without Service Pack 1 ignore a FSCTL_SRV_NOTIFY_TRANSACTION request specifying a valid **FileId** and don't send a response to the client, and reply to a FSCTL_SRV_NOTIFY_TRANSACTION with an invalid or -1 **FileId** with STATUS_INVALID_PARAMETER.

<228> [Section 3.3.5.15.1](#): Windows enumerates previous versions of an existing file by forming a **FileBothDirectoryInformation** object store request for the file including an @GMT-* wildcard as described in [\[MS-SMB\]](#) section 2.2.1.1.1 and [\[MS-FSA\]](#) section 3.1.5.5.4.1.

<229> [Section 3.3.5.15.1](#): Windows-based SMB2 server will place 2 extra bytes set to zero in **Snapshots** array and set **SnapShotArraySize** to 2, if **NumberOfSnapshots** is zero.

<230> [Section 3.3.5.15.2](#): A Windows-based DFS server does not return any data to the caller if the buffer supplied to FSCTL_GET_DFS_REFERRALS is too small.

<231> [Section 3.3.5.15.3](#): Windows-based servers return STATUS_INVALID_DEVICE_REQUEST if the FSCTL_PIPE_TRANSCEIVE being executed is not a named pipe share.

<232> [Section 3.3.5.15.3](#): Windows SMB2 servers send an interim response to the client if the read/write attempt is not finished in 1 millisecond.

<233> [Section 3.3.5.15.3](#): Some Windows-based SMB2 servers return the input buffer that was received in the request as part of the response. Windows 7 and Windows Server 2008 R2 will not return the input buffer that was received in the request, and the **InputCount** field is always zero. Windows Vista and Windows Server 2008 will send back the input buffer based on the **InputOffset** and **InputCount** fields indicated in the request.

<234> [Section 3.3.5.15.3](#): Windows-based SMB2 servers set **OutputOffset** to **InputOffset** + **InputCount**, rounded up to a multiple of 8.

<235> [Section 3.3.5.15.4](#): Windows-based servers return STATUS_INVALID_DEVICE_REQUEST, if FSCTL_PIPE_PEEK request being executed is not a named pipe share.

<236> [Section 3.3.5.15.4](#): Windows SMB2 servers will set **OutputOffset** to **InputOffset** + **InputCount**, rounded up to a multiple of 8.

<237> [Section 3.3.5.15.5](#): Windows servers do not support any additional contexts.

<238> [Section 3.3.5.15.5](#): Windows servers construct the 24-byte blob using **Open.DurableFileId** and other pieces of information which include the process ID of the caller and a timestamp.

<239> [Section 3.3.5.15.6](#): Windows performs server-side copy reads from a file with Server Requests a Read [\[MS-FSA\]](#) section 3.1.5.2.

Object Store parameter	SMB2 parameter
ByteOffset	SourceOffset
ByteCount	Length
IsNonCached	FALSE

<240> [Section 3.3.5.15.6](#): Windows servers will return STATUS_INVALID_VIEW_SIZE instead of STATUS_END_OF_FILE.

<241> [Section 3.3.5.15.6:](#) Windows performs server-side copy writes to a file with Application Requests a Write [\[MS-FSA\]](#) section 3.1.5.3.

Object Store parameter	SMB2 parameter
ByteOffset	TargetOffset
ByteCount	Length
IsWriteThrough	FALSE
IsNonCached	FALSE

<242> [Section 3.3.5.15.7:](#) Windows 7 and Windows Server 2008 R2 servers support the FSCTL_SRV_READ_HASH request.

<243> [Section 3.3.5.15.8:](#) The following FSCTLs are explicitly blocked by Windows-based SMB2 server and not passed through to the object store. They are failed with STATUS_NOT_SUPPORTED.

FSCTL_REQUEST_OPLOCK_LEVEL_1 (0x00090000)

FSCTL_REQUEST_OPLOCK_LEVEL_2 (0x00090004)

FSCTL_REQUEST_BATCH_OPLOCK (0x00090008)

FSCTL_REQUEST_FILTER_OPLOCK (0x0009005C)

FSCTL_OPLOCK_BREAK_ACKNOWLEDGE (0x0009000C)

FSCTL_OPBATCH_ACK_CLOSE_PENDING (0x00090010)

FSCTL_OPLOCK_BREAK_NOTIFY (0x00090014)

FSCTL_MOVE_FILE (0x00090074)

FSCTL_MARK_HANDLE (0x000900FC)

FSCTL_QUERY_RETRIEVAL_POINTERS (0x0009003B)

FSCTL_PIPE_ASSIGN_EVENT (0x00110000)

FSCTL_GET_VOLUME_BITMAP (0x0009006F)

FSCTL_GET_NTFS_FILE_RECORD (0x00090068)

FSCTL_INVALIDATE_VOLUMES (0x00090054)

FSCTL_READ_USN_JOURNAL (0x000900BB)

FSCTL_CREATE_USN_JOURNAL (0x000900E7)

FSCTL_QUERY_USN_JOURNAL (0x000900F4)

FSCTL_DELETE_USN_JOURNAL (0x000900F8)

FSCTL_ENUM_USN_DATA (0x000900B3)

FSCTL_QUERY_DEPENDENT_VOLUME (0x000901F0)

FSCTL_SD_GLOBAL_CHANGE (0x000901F4)
FSCTL_GET_BOOT_AREA_INFO (0x00090230)
FSCTL_GET_RETRIEVAL_POINTER_BASE (0x00090234)
FSCTL_SET_PERSISTENT_VOLUME_STATE (0x00090238)
FSCTL_QUERY_PERSISTENT_VOLUME_STATE (0x0009023C)
FSCTL_REQUEST_OPLOCK (0x00090240)
FSCTL_TXFS_MODIFY_RM (0x00098144)
FSCTL_TXFS_QUERY_RM_INFORMATION (0x00094148)
FSCTL_TXFS_ROLLFORWARD_REDO (0x00098150)
FSCTL_TXFS_ROLLFORWARD_UNDO (0x00098154)
FSCTL_TXFS_START_RM (0x00098158)
FSCTL_TXFS_SHUTDOWN_RM (0x0009815C)
FSCTL_TXFS_READ_BACKUP_INFORMATION (0x00094160)
FSCTL_TXFS_WRITE_BACKUP_INFORMATION (0x00098164)
FSCTL_TXFS_CREATE_SECONDARY_RM (0x00098168)
FSCTL_TXFS_GET_METADATA_INFO (0x0009416C)
FSCTL_TXFS_GET_TRANSACTED_VERSION (0x00094170)
FSCTL_TXFS_SAVEPOINT_INFORMATION (0x00098178)
FSCTL_TXFS_CREATE_MINIVERSION (0x0009817C)
FSCTL_TXFS_TRANSACTION_ACTIVE (0x0009418C)
FSCTL_TXFS_LIST_TRANSACTIONS (0x000941E4)
FSCTL_TXFS_READ_BACKUP_INFORMATION2 (0x000901F8)
FSCTL_TXFS_WRITE_BACKUP_INFORMATION2 (0x00090200)

Windows-based SMB2 servers fail FSCTLs whose transfer type is METHOD_NEITHER with error STATUS_NOT_SUPPORTED except the following ones. For more information about FSCTL transfer type, see [\[MSDN-IoCtlCodes\]](#).

FSCTL_PIPE_TRANSCEIVE (0x0011C017)
FSCTL_QUERY_ALLOCATED_RANGES (0x000940CF)
FSCTL_WRITE_USN_CLOSE_RECORD (0x000900EF)
FSCTL_READ_FILE_USN_DATA (0x000900EB)
FSCTL_GET_RETRIEVAL_POINTERS (0x00090073)

FSCTL_FIND_FILES_BY_SID (0x0009008F)

FSCTL_SRV_READ_HASH (0x001441BB)

<244> [Section 3.3.5.15.8](#): Windows performs passthrough FSCTL operations via Server Requests an FsControl Request [\[MS-FSA\]](#) section 3.1.5.9.

<245> [Section 3.3.5.15.8](#): Windows-based SMB2 servers will set **OutputOffset** to **InputOffset + InputCount**, rounded up to a multiple of 8.

<246> [Section 3.3.5.15.9](#): Windows 7 and Windows Server 2008 R2 servers support the FSCTL_LMR_REQUEST_RESILIENCY request.

<247> [Section 3.3.5.16](#): Windows performs cancellation of in-progress requests via the interface in [\[MS-FSA\]](#) section 3.1.5.19, Server Requests Canceling an Operation, passing **Request.CancelRequestId** as an input parameter.

<248> [Section 3.3.5.18](#): The Windows SMB2 server implementation closes and reopens the directory handle in order to "reset" the enumeration state. So any outstanding operations on the directory handle will be failed with a STATUS_FILE_CLOSED error.

<249> [Section 3.3.5.18](#): If the length of the received data is less than the size of SMB2 header (0x40) plus size of SMB2 QUERY_DIRECTORY request (0x21), Windows servers fail the request with STATUS_INVALID_PARAMETER. Otherwise, if **FileNameLength** is 0 and the underlying file system is **NTFS**, Windows servers fail the request with STATUS_OBJECT_NAME_INVALID.

<250> [Section 3.3.5.18](#): Windows-based servers do not support resuming an enumeration at a specified **FileIndex**. The server will ignore this flag.

<251> [Section 3.3.5.18](#): Windows performs directory query information requests via the corresponding interfaces in [\[MS-FSA\]](#) section 3.1.5.5.4:

- FileBothDirectoryInformation: [\[MS-FSA\]](#) section 3.1.5.5.4.1.
- FileDirectoryInformation: [\[MS-FSA\]](#) section 3.1.5.5.4.2.
- FileFullDirectoryInformation: [\[MS-FSA\]](#) section 3.1.5.5.4.3.
- FileIdBothDirectoryInformation: [\[MS-FSA\]](#) section 3.1.5.5.4.4.
- FileIdFullDirectoryInformation: [\[MS-FSA\]](#) section 3.1.5.5.4.5.
- FileNamesInformation: [\[MS-FSA\]](#) section 3.1.5.5.4.6.

<252> [Section 3.3.5.19](#): Windows Vista and Windows Server 2008 limit **OutputBufferLength** size to 256 KB.

<253> [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 IOCTL Request](#) for FSCTL_SRV_COPYCHUNK or FSCTL_SRV_COPYCHUNK_WRITE (section 2.2.31) when oplock break happens, [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 milliseconds, [SMB2 Write Request \(section 2.2.21\)](#) for large file write, [SMB2 lock Request \(section 2.2.26\)](#) if the SMB2_LOCKFLAG_FAIL_IMMEDIATELY flag is not set, and [SMB2 FLUSH Request \(section 2.2.17\)](#) for named pipes.

<254> [Section 3.3.5.19](#): Windows requests ChangeNotify processing via Server Requests Change Notifications for a Directory in [\[MS-FSA\]](#) section 3.1.5.10. If the SMB2_WATCH_TREE flag is set, the WatchTree boolean is passed as TRUE. ChangeNotify notification is reported as described in [\[MS-FSA\]](#) section 3.1.5.10.1.

<255> [Section 3.3.5.20.1](#): Windows-based SMB2 servers fail the following request levels with STATUS_NOT_SUPPORTED instead of STATUS_INVALID_INFO_CLASS: 41, 42, 43, 47, 49, 51, 52, and 53.

<256> [Section 3.3.5.20.1](#): Windows-based SMB2 servers will set the **CurrentByteOffset** to any value.

<257> [Section 3.3.5.20.1](#): Windows performs SMB2 GET_INFO SMB2_0_INFO_FILE processing via the subsection of [\[MS-FSA\]](#) 3.1.5.11 corresponding to the requested FILE_INFORMATION_CLASS value of the **FileInfoClass** request field, as listed in section [2.2.37](#).

<258> [Section 3.3.5.20.2](#): Windows performs SMB2 GET_INFO SMB2_0_INFO_FILESYSTEM processing via the subsection of [\[MS-FSA\]](#) 3.1.5.11 corresponding to the requested FS_INFORMATION_CLASS value of the **FileInfoClass** request field, as listed in section [2.2.37](#).

<259> [Section 3.3.5.20.2](#): SetFsInfo calls to Windows servers fail with STATUS_ACCESS_DENIED because Windows servers do not allow setting volume information over the network.

<260> [Section 3.3.5.20.3](#): Windows performs SMB2 GET_INFO SMB2_0_INFO_SECURITY processing via Server Requests a Query of Security Information ([\[MS-FSA\]](#) section 3.1.5.13).

<261> [Section 3.3.5.20.4](#): Windows-based servers do support quotas, if configured.

<262> [Section 3.3.5.20.4](#): Windows performs SMB2 GET_INFO SMB2_0_INFO_QUOTA processing via Server Requests a Query of Quota Information ([\[MS-FSA\]](#) section 3.1.5.20).

<263> [Section 3.3.5.21.1](#): Windows-based SMB2 servers fail the following request levels with STATUS_NOT_SUPPORTED instead of STATUS_INVALID_INFO_CLASS: 30, 41, 42, 43.

<264> [Section 3.3.5.21.1](#): Windows performs SMB2 SET_INFO SMB2_0_INFO_FILE processing via the subsection of [\[MS-FSA\]](#) section 3.1.5.14 corresponding to the requested FILE_INFORMATION_CLASS value of the **FileInfoClass** request field, as listed in section [2.2.37](#).

<265> [Section 3.3.5.21.2](#): Windows performs SMB2 SET_INFO SMB2_0_INFO_FILESYSTEM processing via the subsection of [\[MS-FSA\]](#) section 3.1.5.15 corresponding to the requested FS_INFORMATION_CLASS value of the **FileInfoClass** request field, as listed in section [2.2.37](#).

<266> [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 returns STATUS_SUCCESS.

<267> [Section 3.3.5.21.3](#): Windows performs SMB2 SET_INFO SMB2_0_INFO_SECURITY processing via Server Requests Setting of Security Information [\[MS-FSA\]](#) section 3.1.5.16.

<268> [Section 3.3.5.21.4](#): Windows servers do support quotas, if configured.

<269> [Section 3.3.5.21.4](#): Windows performs SMB2 SET_INFO SMB2_0_INFO_QUOTA processing via Server Requests Setting of Quota Information ([\[MS-FSA\]](#) section 3.1.5.21).

<270> [Section 3.3.7.1](#): Windows-based SMB2 servers provide a statically configured durable handle time-out, which defaults to 16 minutes.

[<271> Section 3.3.7.1:](#) Windows performs cancellation of in-progress requests via the interface in [\[MS-FSA\]](#) section 3.1.5.19, Server Requests Canceling an Operation, passing **Request.CancelRequestId** as an input parameter.

7 Change Tracking

This section identifies changes that were made to the [MS-SMB2] protocol document between the May 2011 and June 2011 releases. Changes are classified as New, Major, Minor, Editorial, or No change.

The revision class **New** means that a new document is being released.

The revision class **Major** means that the technical content in the document was significantly revised. Major changes affect protocol interoperability or implementation. Examples of major changes are:

- A document revision that incorporates changes to interoperability requirements or functionality.
- An extensive rewrite, addition, or deletion of major portions of content.
- The removal of a document from the documentation set.
- Changes made for template compliance.

The revision class **Minor** means that the meaning of the technical content was clarified. Minor changes do not affect protocol interoperability or implementation. Examples of minor changes are updates to clarify ambiguity at the sentence, paragraph, or table level.

The revision class **Editorial** means that the language and formatting in the technical content was changed. Editorial changes apply to grammatical, formatting, and style issues.

The revision class **No change** means that no new technical or language changes were introduced. The technical content of the document is identical to the last released version, but minor editorial and formatting changes, as well as updates to the header and footer information, and to the revision summary, may have been made.

Major and minor changes can be described further using the following change types:

- New content added.
- Content updated.
- Content removed.
- New product behavior note added.
- Product behavior note updated.
- Product behavior note removed.
- New protocol syntax added.
- Protocol syntax updated.
- Protocol syntax removed.
- New content added due to protocol revision.
- Content updated due to protocol revision.
- Content removed due to protocol revision.
- New protocol syntax added due to protocol revision.

- Protocol syntax updated due to protocol revision.
- Protocol syntax removed due to protocol revision.
- New content added for template compliance.
- Content updated for template compliance.
- Content removed for template compliance.
- Obsolete document removed.

Editorial changes are always classified with the change type **Editorially updated**.

Some important terms used in the change type descriptions are defined as follows:

- **Protocol syntax** refers to data elements (such as packets, structures, enumerations, and methods) as well as interfaces.
- **Protocol revision** refers to changes made to a protocol that affect the bits that are sent over the wire.

The changes made to this document are listed in the following table. For more information, please contact protocol@microsoft.com.

Section	Tracking number (if applicable) and description	Major change (Y or N)	Change type
1.2 References	Added explanatory statement regarding the removal of the publishing year from Microsoft Open Specification document references.	N	Content updated.
3.2.4.1.5 Sending Multi-Credit Requests	64159 Updated the number of credits sent by Windows clients.	N	Product behavior note updated.
3.3.4.7 Object Store Indicates a Lease Break	64157 Updated the processing rules for setting lease acknowledgment.	N	Content updated.
3.3.4.7 Object Store Indicates a Lease Break	64140 Updated the processing rules for cases in which a lease entry is found.	N	Content updated.
3.3.5.2.4 Verifying the Credit Charge and the Payload Size	64098 Added a new product behavior note for verifying the payload of a request.	Y	New product behavior note added.

8 Index

A

Abstract data model
 client ([section 3.1.1](#) 110, [section 3.2.1](#) 111)
 server ([section 3.1.1](#) 110, [section 3.3.1](#) 175)
[Applicability](#) 20
[Application Requests Reauthenticating a User](#) 123
[Authenticating the user](#) 122

C

[Capability negotiation](#) 20
[Change notifications algorithm](#) 177
[Change tracking](#) 334
Client
 abstract data model ([section 3.1.1](#) 110, [section 3.2.1](#) 111)
 [global connections](#) 111
 higher-layer triggered events
 [overview](#) 115
 [re-establishing a durable open](#) 128
 [requesting applying of file attributes](#) 133
 [requesting applying of file security attributes](#) 137
 [requesting applying of file system attributes](#) 135
 [requesting applying of quota information](#) 139
 [requesting cancellation of operation](#) 158
 [requesting change of notifications for directory](#) 142
 [requesting closing of file or named pipe](#) 129
 [requesting closing of share connection](#) 158
 [requesting connection to share](#) 118
 [requesting enumeration of directory](#) 141
 [requesting flushing of cached data](#) 140
 [requesting IO control code operation](#) 144
 [requesting locking of array of byte ranges](#) 143
 [requesting opening of file](#) 124
 [requesting querying for file attributes](#) 131
 [requesting querying for file security attributes](#) 136
 [requesting querying for file system attributes](#) 134
 [requesting querying for quota information](#) 137
 [requesting reading from file or named pipe](#) 129
 [requesting session key for authenticated context](#) 159
 [requesting termination of authenticated context](#) 158
 [requesting unlocking of array of byte ranges](#) 156
 [requesting writing to file or named pipe](#) 130
 [sending any outgoing message](#) 115
 [signing outgoing message](#) 110
 initialization ([section 3.1.3](#) 110, [section 3.2.3](#) 115)
 local events ([section 3.1.7](#) 111, [section 3.2.7](#) 175)

message processing
 [overview](#) 159
 [receiving any message](#) 159
 [receiving SMB2 CHANGE_NOTIFY response](#) 171
 [receiving SMB2 CLOSE response](#) 168
 [receiving SMB2 CREATE response for new create operation](#) 165
 [receiving SMB2 CREATE response for open reestablishment](#) 167
 [receiving SMB2 FLUSH response](#) 168
 [receiving SMB2 IOCTL response](#) 169
 [receiving SMB2 LOCK response](#) 168
 [receiving SMB2 LOGOFF response](#) 164
 [receiving SMB2 NEGOTIATE response](#) 161
 [receiving SMB2 OPLOCK_BREAK notification](#) 172
 [receiving SMB2 QUERY_DIRECTORY response](#) 171
 [receiving SMB2 QUERY_INFO response](#) 171
 [receiving SMB2 READ response](#) 168
 [receiving SMB2 SESSION_SETUP response](#) 162
 [receiving SMB2 SET_INFO response](#) 172
 [receiving SMB2 TREE_CONNECT response](#) 164
 [receiving SMB2 TREE_DISCONNECT response](#) 165
 [receiving SMB2 WRITE response](#) 168
 [verifying incoming message](#) 111
[message sequence numbers algorithm](#) 118
[per open](#) 114
[per session](#) 113
[per SMB2 transport connection](#) 112
[per tree connect](#) 113
[per unique open file](#) 113
[required global data](#) 110
sequencing rules
 [overview](#) 159
 [receiving any message](#) 159
 [receiving SMB2 CHANGE_NOTIFY response](#) 171
 [receiving SMB2 CLOSE response](#) 168
 [receiving SMB2 CREATE response for new create operation](#) 165
 [receiving SMB2 CREATE response for open reestablishment](#) 167
 [receiving SMB2 FLUSH response](#) 168
 [receiving SMB2 IOCTL response](#) 169
 [receiving SMB2 LOCK response](#) 168
 [receiving SMB2 LOGOFF response](#) 164
 [receiving SMB2 NEGOTIATE response](#) 161
 [receiving SMB2 OPLOCK_BREAK notification](#) 172
 [receiving SMB2 QUERY_DIRECTORY response](#) 171
 [receiving SMB2 QUERY_INFO response](#) 171
 [receiving SMB2 READ response](#) 168
 [receiving SMB2 SESSION_SETUP response](#) 162
 [receiving SMB2 SET_INFO response](#) 172
 [receiving SMB2 TREE_CONNECT response](#) 164

- [receiving SMB2 TREE_DISCONNECT response](#) 165
- [receiving SMB2 WRITE response](#) 168
- [verifying incoming message](#) 111
- timer events ([section 3.1.6](#) 111, [section 3.2.6](#) 174)
- timers ([section 3.1.2](#) 110, [section 3.2.2](#) 115)
- [Connecting to the share](#) 124
- [Connecting to the target server](#) 120
- [Connections - global](#) 111
- [Credit granting algorithm](#) 176

D

- [Data - global](#) 110
- Data model – abstract
 - client ([section 3.1.1](#) 110, [section 3.2.1](#) 111)
 - server ([section 3.1.1](#) 110, [section 3.3.1](#) 175)
- [Directory Access Mask packet](#) 52
- [Disconnecting example](#) 307
- [Durable open scavenger timer](#) 186
- [Durable open scavenger timer event](#) 262

E

- Example
 - [disconnecting](#) 307
 - [logging off](#) 307
- Examples
 - [multi-protocol negotiate](#) 265
 - [named pipe](#) 282
 - [negotiating SMB 2.10 dialect by using multi-protocol negotiate](#) 271
 - [overview](#) 265
 - remote files
 - [reading](#) 291
 - [writing](#) 297
 - [SMB2 negotiate](#) 276

F

- [Fields – vendor-extensible](#) 21
- [FILE_NOTIFY_INFORMATION packet](#) 99
- [File Pipe Printer Access Mask packet](#) 51

G

- Global
 - [connections](#) 111
 - [data](#) 110
 - [structures](#) 178
- [Glossary](#) 12

H

- [HASH_HEADER packet](#) 88
- Higher-layer triggered events
 - client
 - [overview](#) 115
 - [re-establishing a durable open](#) 128
 - [requesting applying of file attributes](#) 133

- [requesting applying of file security attributes](#) 137
- [requesting applying of file system attributes](#) 135
- [requesting applying of quota information](#) 139
- [requesting cancellation of operation](#) 158
- [requesting change of notifications for directory](#) 142
- [requesting closing of file or named pipe](#) 129
- [requesting closing of share connection](#) 158
- [requesting connection to share](#) 118
- [requesting enumeration of directory](#) 141
- [requesting flushing of cached data](#) 140
- [requesting IO control code operation](#) 144
- [requesting locking of array of byte ranges](#) 143
- [requesting opening of file](#) 124
- [requesting querying for file attributes](#) 131
- [requesting querying for file security attributes](#) 136
- [requesting querying for file system attributes](#) 134
- [requesting querying for quota information](#) 137
- [requesting reading from file or named pipe](#) 129
- [requesting session key for authenticated context](#) 159
- [requesting termination of authenticated context](#) 158
- [requesting unlocking of array of byte ranges](#) 156
- [requesting writing to file or named pipe](#) 130
- [sending any outgoing message](#) 115
- [signing outgoing message](#) 110
- server
 - [notification that DFS is active](#) 192
 - [notification that share is DFS share](#) 192
 - [notification that share is not DFS share](#) 192
 - [object store indicating lease break](#) 191
 - [object store indicating oplock break](#) 191
 - [overview](#) 187
 - [requesting security context](#) 192
 - [requesting session key](#) 190
 - [sending any outgoing message](#) 187
 - [sending error response](#) 190
 - [sending interim response for asynchronous operation](#) 188
 - [signing outgoing message](#) 110

I

- [Idle connection timer](#) 115
- [Idle connection timer event](#) 174
- [Implementer - security considerations](#) 310
- [Incoming message - verifying](#) 111
- [Index of security parameters](#) 310
- [Informative references](#) 16
- Initialization
 - client ([section 3.1.3](#) 110, [section 3.2.3](#) 115)
 - server ([section 3.1.3](#) 110, [section 3.3.3](#) 186)
- [Introduction](#) 12

L

[Leasing algorithm](#) 177

Local events

client ([section 3.1.7](#) 111, [section 3.2.7](#) 175)
server ([section 3.1.7](#) 111, [section 3.3.7](#) 263)

[Logging off example](#) 307

M

Message processing

client

[overview](#) 159

[receiving any message](#) 159

[receiving SMB2 CHANGE_NOTIFY response](#) 171

[receiving SMB2 CLOSE response](#) 168

[receiving SMB2 CREATE response for new
create operation](#) 165

[receiving SMB2 CREATE response for open
reestablishment](#) 167

[receiving SMB2 FLUSH response](#) 168

[receiving SMB2 IOCTL response](#) 169

[receiving SMB2 LOCK response](#) 168

[receiving SMB2 LOGOFF response](#) 164

[receiving SMB2 NEGOTIATE response](#) 161

[receiving SMB2 OPLOCK_BREAK notification](#)
172

[receiving SMB2 QUERY_DIRECTORY response](#)
171

[receiving SMB2 QUERY_INFO response](#) 171

[receiving SMB2 READ response](#) 168

[receiving SMB2 SESSION_SETUP response](#) 162

[receiving SMB2 SET_INFO response](#) 172

[receiving SMB2 TREE_CONNECT response](#) 164

[receiving SMB2 TREE_DISCONNECT response](#)
165

[receiving SMB2 WRITE response](#) 168

[verifying incoming message](#) 111

server

[accepting incoming connection](#) 200

[overview](#) 200

[receiving any message](#) 201

[receiving SMB_COM_NEGOTIATE](#) 204

[receiving SMB2 CANCEL request](#) 246

[receiving SMB2 CHANGE_NOTIFY request](#) 249

[receiving SMB2 CLOSE request](#) 226

[receiving SMB2 CREATE request](#) 216

[receiving SMB2 ECHO request](#) 246

[receiving SMB2 FLUSH request](#) 227

[receiving SMB2 IOCTL request](#) 234

[receiving SMB2 LOCK request](#) 231

[receiving SMB2 LOGOFF request](#) 213

[receiving SMB2 NEGOTIATE request](#) 206

[receiving SMB2 OPLOCK_BREAK](#)

[acknowledgment](#) 259

[receiving SMB2 QUERY_DIRECTORY request](#)
246

[receiving SMB2 QUERY_INFO request](#) 251

[receiving SMB2 READ request](#) 228

[receiving SMB2 SESSION_SETUP request](#) 208

[receiving SMB2 SET_INFO request](#) 256

[receiving SMB2 TREE_CONNECT request](#) 213

[receiving SMB2 TREE_DISCONNECT request](#)
215

[receiving SMB2 WRITE request](#) 230

[verifying incoming message](#) 111

Message sequence numbers algorithm ([section 3.2.4.1.6](#) 118, [section 3.3.1.1](#) 176)

Messages

[overview](#) 23

[signing outgoing](#) 110

[SMB2 Packet Header](#) 24

[syntax](#) 23

[transport](#) 23

[verifying incoming](#) 111

[Multi-protocol negotiate example](#) 265

N

[Named pipe example](#) 282

[Negotiating SMB 2.10 dialect by using multi-
protocol negotiate example](#) 271

[Negotiating the protocol](#) 121

[Network disconnect](#) 175

[NETWORK_RESILIENCY_REQUEST Request packet](#)
90

[Normative references](#) 15

O

[Oplock break acknowledgment timer](#) 185

[Oplock break acknowledgment timer event](#) 262

[Outgoing message - signing](#) 110

[Overview \(synopsis\)](#) 16

P

[Parameters - security index](#) 310

[Per lease structures](#) 184

[Per lease table structures](#) 184

[Per open](#) 114

[Per open structures](#) 182

[Per session](#) 113

[Per session structures](#) 181

[Per share structures](#) 179

[Per SMB2 transport connection](#) 112

[Per transport connection structures](#) 181

[Per tree connect](#) 113

[Per tree connect structures](#) 182

[Per unique open file](#) 113

[Pipe - named - example](#) 282

[Preconditions](#) 19

[Prerequisites](#) 19

[Product behavior](#) 311

R

References

[informative](#) 16

[normative](#) 15

[Relationship to other protocols](#) 18

Remote files

[reading - example](#) 291

[writing - example](#) 297

[Request expiration timer](#) 115

[Request expiration timer event](#) 174

[Resilient open scavenger timer](#) 186
[Resilient open scavenger timer event](#) 262

S

Security

[implementer considerations](#) 310
[overview](#) 310
[parameter index](#) 310

Sequencing rules

client

[overview](#) 159
[receiving any message](#) 159
[receiving SMB2 CHANGE_NOTIFY response](#) 171
[receiving SMB2 CLOSE response](#) 168
[receiving SMB2 CREATE response for new create operation](#) 165
[receiving SMB2 CREATE response for open reestablishment](#) 167
[receiving SMB2 FLUSH response](#) 168
[receiving SMB2 IOCTL response](#) 169
[receiving SMB2 LOCK response](#) 168
[receiving SMB2 LOGOFF response](#) 164
[receiving SMB2 NEGOTIATE response](#) 161
[receiving SMB2 OPLOCK_BREAK notification](#) 172
[receiving SMB2 QUERY_DIRECTORY response](#) 171
[receiving SMB2 QUERY_INFO response](#) 171
[receiving SMB2 READ response](#) 168
[receiving SMB2 SESSION_SETUP response](#) 162
[receiving SMB2 SET_INFO response](#) 172
[receiving SMB2 TREE_CONNECT response](#) 164
[receiving SMB2 TREE_DISCONNECT response](#) 165
[receiving SMB2 WRITE response](#) 168
[verifying incoming message](#) 111

server

[accepting incoming connection](#) 200
[overview](#) 200
[receiving any message](#) 201
[receiving SMB_COM_NEGOTIATE](#) 204
[receiving SMB2 CANCEL request](#) 246
[receiving SMB2 CHANGE_NOTIFY request](#) 249
[receiving SMB2 CLOSE request](#) 226
[receiving SMB2 CREATE request](#) 216
[receiving SMB2 ECHO request](#) 246
[receiving SMB2 FLUSH request](#) 227
[receiving SMB2 IOCTL request](#) 234
[receiving SMB2 LOCK request](#) 231
[receiving SMB2 LOGOFF request](#) 213
[receiving SMB2 NEGOTIATE request](#) 206
[receiving SMB2 OPLOCK_BREAK acknowledgment](#) 259
[receiving SMB2 QUERY_DIRECTORY request](#) 246
[receiving SMB2 QUERY_INFO request](#) 251
[receiving SMB2 READ request](#) 228
[receiving SMB2 SESSION_SETUP request](#) 208
[receiving SMB2 SET_INFO request](#) 256
[receiving SMB2 TREE_CONNECT request](#) 213

[receiving SMB2 TREE_DISCONNECT request](#) 215
[receiving SMB2 WRITE request](#) 230
[verifying incoming message](#) 111

Server

[abstract data model](#) ([section 3.1.1](#) 110, [section 3.3.1](#) 175)
[change notifications algorithm](#) 177
[credit granting algorithm](#) 176
[global structures](#) 178
[higher-layer triggered events](#)
[notification that DFS is active](#) 192
[notification that share is DFS share](#) 192
[notification that share is not DFS share](#) 192
[object store indicating lease break](#) 191
[object store indicating oplock break](#) 191
[overview](#) 187
[requesting security context](#) 192
[requesting session key](#) 190
[sending any outgoing message](#) 187
[sending error response](#) 190
[sending interim response for asynchronous operation](#) 188
[signing outgoing message](#) 110
[initialization](#) ([section 3.1.3](#) 110, [section 3.3.3](#) 186)
[leasing algorithm](#) 177
[local events](#) ([section 3.1.7](#) 111, [section 3.3.7](#) 263)
[message processing](#)
[accepting incoming connection](#) 200
[overview](#) 200
[receiving any message](#) 201
[receiving SMB_COM_NEGOTIATE](#) 204
[receiving SMB2 CANCEL request](#) 246
[receiving SMB2 CHANGE_NOTIFY request](#) 249
[receiving SMB2 CLOSE request](#) 226
[receiving SMB2 CREATE request](#) 216
[receiving SMB2 ECHO request](#) 246
[receiving SMB2 FLUSH request](#) 227
[receiving SMB2 IOCTL request](#) 234
[receiving SMB2 LOCK request](#) 231
[receiving SMB2 LOGOFF request](#) 213
[receiving SMB2 NEGOTIATE request](#) 206
[receiving SMB2 OPLOCK_BREAK acknowledgment](#) 259
[receiving SMB2 QUERY_DIRECTORY request](#) 246
[receiving SMB2 QUERY_INFO request](#) 251
[receiving SMB2 READ request](#) 228
[receiving SMB2 SESSION_SETUP request](#) 208
[receiving SMB2 SET_INFO request](#) 256
[receiving SMB2 TREE_CONNECT request](#) 213
[receiving SMB2 TREE_DISCONNECT request](#) 215
[receiving SMB2 WRITE request](#) 230
[verifying incoming message](#) 111
[message sequence numbers algorithm](#) 176
[per lease structures](#) 184
[per lease table structures](#) 184
[per open structures](#) 182

per session structures	181	SMB2 CREATE QUERY MAXIMAL ACCESS RESPON	
per share structures	179	SE packet	63
per transport connection structures	181	SMB2 CREATE QUERY ON DISK ID	59
per tree connect structures	182	SMB2 CREATE QUERY ON DISK ID packet	64
required global data	110	SMB2 CREATE Request packet	45
sequencing rules		SMB2 CREATE REQUEST LEASE packet	58
accepting incoming connection	200	SMB2 CREATE Response packet	59
overview	200	SMB2 CREATE RESPONSE LEASE packet	64
receiving any message	201	SMB2 CREATE SD BUFFER	62
receiving SMB COM NEGOTIATE	204	SMB2 CREATE TIMEWARP TOKEN	64
receiving SMB2 CANCEL request	246	SMB2 CREATE TIMEWARP TOKEN packet	58
receiving SMB2 CHANGE_NOTIFY request	249	SMB2 ECHO Request packet	83
receiving SMB2 CLOSE request	226	SMB2 ECHO Response packet	83
receiving SMB2 CREATE request	216	SMB2 ERROR Response packet	31
receiving SMB2 ECHO request	246	SMB2 FILEID packet	62
receiving SMB2 FLUSH request	227	SMB2 FLUSH Request packet	68
receiving SMB2 IOCTL request	234	SMB2 FLUSH Response packet	69
receiving SMB2 LOCK request	231	SMB2 IOCTL Request packet	84
receiving SMB2 LOGOFF request	213	SMB2 IOCTL Response packet	90
receiving SMB2 NEGOTIATE request	206	SMB2 Lease Break Acknowledgment packet	77
receiving SMB2 OPLOCK BREAK		SMB2 Lease Break Notification packet	75
acknowledgment	259	SMB2 Lease Break Response packet	79
receiving SMB2 QUERY DIRECTORY request		SMB2 LOCK ELEMENT packet	81
	246	SMB2 LOCK Request packet	80
receiving SMB2 QUERY INFO request	251	SMB2 LOCK Response packet	82
receiving SMB2 READ request	228	SMB2 LOGOFF Request packet	41
receiving SMB2 SESSION_SETUP request	208	SMB2 LOGOFF Response packet	42
receiving SMB2 SET_INFO request	256	SMB2 NEGOTIATE Request packet	35
receiving SMB2 TREE_CONNECT request	213	SMB2 NEGOTIATE Response packet	36
receiving SMB2 TREE_DISCONNECT request	215	SMB2 Oplock Break Acknowledgment packet	76
receiving SMB2 WRITE request	230	SMB2 Oplock Break Notification packet	74
verifying incoming message	111	SMB2 Oplock Break Response packet	78
timer events (section 3.1.6 111, section 3.3.6 262)		SMB2 Packet Header ASYNC packet	24
timers (section 3.1.2 110, section 3.3.2 185)		SMB2 Packet Header SYNC packet	27
Session expiration timer	186	SMB2 QUERY DIRECTORY Request packet	94
Session expiration timer event	262	SMB2 QUERY DIRECTORY Response packet	96
SMB2 LOCK Request packet	80	SMB2 QUERY INFO Request packet	100
SMB2 negotiate example	276	SMB2 QUERY INFO Response packet	105
SMB2 Packet Header	24	SMB2 QUERY QUOTA INFO packet	104
SMB2 CANCEL Request packet	83	SMB2 READ Request packet	69
SMB2 CHANGE_NOTIFY Request packet	97	SMB2 READ Response packet	70
SMB2 CHANGE_NOTIFY Response packet	99	SMB2 SESSION_SETUP Request packet	39
SMB2 CLOSE Request packet	65	SMB2 SESSION_SETUP Response packet	40
SMB2 CLOSE Response packet	66	SMB2 SET_INFO Request packet	106
SMB2 CREATE_ALLOCATION_SIZE	63	SMB2 SET_INFO Response packet	109
SMB2 CREATE_ALLOCATION_SIZE packet	57	SMB2 TREE_CONNECT Request packet	42
SMB2 CREATE_CONTEXT Response Values	62	SMB2 TREE_CONNECT Response packet	43
SMB2 CREATE_CONTEXT Request Values packet		SMB2 TREE_DISCONNECT Request packet	45
	54	SMB2 TREE_DISCONNECT Response packet	45
SMB2 CREATE DURABLE_HANDLE_RECONNECT	63	SMB2 WRITE Request packet	71
SMB2 CREATE DURABLE_HANDLE_RECONNECT packet	56	SMB2 WRITE Response packet	73
SMB2 CREATE DURABLE_HANDLE_REQUEST packet	56	SRV_COPYCHUNK packet	87
SMB2 CREATE DURABLE_HANDLE_RESPONSE packet	63	SRV_COPYCHUNK COPY packet	86
SMB2_CREATE_EA_BUFFER	62	SRV_COPYCHUNK RESPONSE packet	92
SMB2_CREATE_QUERY_MAXIMAL_ACCESS_REQUEST packet	57	SRV_READ_HASH packet	87
		SRV_READ_HASH Response packet	94
		SRV_REQUEST_RESUME_KEY Response packet	93
		SRV_SNAPSHOT_ARRAY packet	92
		Standards assignments	22
		Symbolic Link Error Response packet	31
		Syntax – message	23

T

Timer events

- client ([section 3.1.6](#) 111, [section 3.2.6](#) 174)
- server ([section 3.1.6](#) 111, [section 3.3.6](#) 262)

Timers

- client ([section 3.1.2](#) 110, [section 3.2.2](#) 115)
- server ([section 3.1.2](#) 110, [section 3.3.2](#) 185)

[Tracking changes](#) 334

Transport

- [disconnect](#) 263
- [message](#) 23

Triggered events – higher layer

client

- [overview](#) 115
- [re-establishing a durable open](#) 128
- [requesting applying of file attributes](#) 133
- [requesting applying of file security attributes](#) 137
- [requesting applying of file system attributes](#) 135
- [requesting applying of quota information](#) 139
- [requesting cancellation of operation](#) 158
- [requesting change of notifications for directory](#) 142
- [requesting closing of file or named pipe](#) 129
- [requesting closing of share connection](#) 158
- [requesting connection to share](#) 118
- [requesting enumeration of directory](#) 141
- [requesting flushing of cached data](#) 140
- [requesting IO control code operation](#) 144
- [requesting locking of array of byte ranges](#) 143
- [requesting opening of file](#) 124
- [requesting querying for file attributes](#) 131
- [requesting querying for file security attributes](#) 136
- [requesting querying for file system attributes](#) 134
- [requesting querying for quota information](#) 137
- [requesting reading from file or named pipe](#) 129
- [requesting session key for authenticated context](#) 159
- [requesting termination of authenticated context](#) 158
- [requesting unlocking of array of byte ranges](#) 156
- [requesting writing to file or named pipe](#) 130
- [sending any outgoing message](#) 115
- [signing outgoing message](#) 110

server

- [notification that DFS is active](#) 192
- [notification that share is DFS share](#) 192
- [notification that share is not DFS share](#) 192
- [object store indicating lease break](#) 191
- [object store indicating oplock break](#) 191
- [overview](#) 187
- [requesting security context](#) 192
- [requesting session key](#) 190
- [sending any outgoing message](#) 187
- [sending error response](#) 190

- [sending interim response for asynchronous operation](#) 188
- [signing outgoing message](#) 110

V

[Vendor-extensible fields](#) 21

[Versioning](#) 20