

[MS-SMB2]: Server Message Block (SMB) Protocol Versions 2 and 3

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, email 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
10/22/2006	0.01		MCPP Milestone 1 Initial Availability
01/19/2007	1.0		MCPP Milestone 1
03/02/2007	1.1		Monthly release
04/03/2007	1.2		Monthly release
05/11/2007	1.3		Monthly release
06/01/2007	1.3.1	Editorial	Revised and edited the technical content.
07/03/2007	2.0	Major	MLonghorn+90
07/20/2007	3.0	Major	Updated and revised the technical content.
08/10/2007	4.0	Major	Updated and revised the technical content.
09/28/2007	5.0	Major	Updated and revised the technical content.
10/23/2007	6.0	Major	Updated and revised the technical content.
11/30/2007	7.0	Major	Updated and revised the technical content.
01/25/2008	7.0.1	Editorial	Revised and edited the technical content.
03/14/2008	8.0	Major	Updated and revised the technical content.
05/16/2008	9.0	Major	Updated and revised the technical content.
06/20/2008	10.0	Major	Updated and revised the technical content.
07/25/2008	11.0	Major	Updated and revised the technical content.
08/29/2008	12.0	Major	Updated and revised the technical content.
10/24/2008	13.0	Major	Updated and revised the technical content.
12/05/2008	14.0	Major	Updated and revised the technical content.
01/16/2009	15.0	Major	Updated and revised the technical content.
02/27/2009	16.0	Major	Updated and revised the technical content.
04/10/2009	17.0	Major	Updated and revised the technical content.
05/22/2009	18.0	Major	Updated and revised the technical content.
07/02/2009	19.0	Major	Updated and revised the technical content.
08/14/2009	20.0	Major	Updated and revised the technical content.
09/25/2009	21.0	Major	Updated and revised the technical content.

Date	Revision History	Revision Class	Comments
11/06/2009	22.0	Major	Updated and revised the technical content.
12/18/2009	23.0	Major	Updated and revised the technical content.
01/29/2010	24.0	Major	Updated and revised the technical content.
03/12/2010	25.0	Major	Updated and revised the technical content.
04/23/2010	26.0	Major	Updated and revised the technical content.
06/04/2010	27.0	Major	Updated and revised the technical content.
07/16/2010	28.0	Major	Significantly changed the technical content.
08/27/2010	29.0	Major	Significantly changed the technical content.
10/08/2010	30.0	Major	Significantly changed the technical content.
11/19/2010	31.0	Major	Significantly changed the technical content.
01/07/2011	32.0	Major	Significantly changed the technical content.
02/11/2011	33.0	Major	Significantly changed the technical content.
03/25/2011	34.0	Major	Significantly changed the technical content.
05/06/2011	35.0	Major	Significantly changed the technical content.
06/17/2011	36.0	Major	Significantly changed the technical content.
09/23/2011	37.0	Major	Significantly changed the technical content.
12/16/2011	38.0	Major	Significantly changed the technical content.
03/30/2012	39.0	Major	Significantly changed the technical content.
07/12/2012	40.0	Major	Significantly changed the technical content.
10/25/2012	41.0	Major	Significantly changed the technical content.

Contents

1 Introduction	13
1.1 Glossary	13
1.2 References	16
1.2.1 Normative References	16
1.2.2 Informative References	17
1.3 Overview	18
1.4 Relationship to Other Protocols	20
1.5 Prerequisites/Preconditions	21
1.6 Applicability Statement	22
1.7 Versioning and Capability Negotiation	22
1.8 Vendor-Extensible Fields	24
1.9 Standards Assignments	24
2 Messages.....	25
2.1 Transport	25
2.2 Message Syntax	25
2.2.1 SMB2 Packet Header	26
2.2.1.1 SMB2 Packet Header - ASYNC	27
2.2.1.2 SMB2 Packet Header - SYNC	30
2.2.2 SMB2 ERROR Response Packet	34
2.2.2.1 Symbolic Link Error Response	34
2.2.2.1.1 Handling the Symbolic Link Error Response	36
2.2.3 SMB2 NEGOTIATE Request	38
2.2.4 SMB2 NEGOTIATE Response	40
2.2.5 SMB2 SESSION_SETUP Request	42
2.2.6 SMB2 SESSION_SETUP Response	44
2.2.7 SMB2 LOGOFF Request	45
2.2.8 SMB2 LOGOFF Response	45
2.2.9 SMB2 TREE_CONNECT Request	46
2.2.10 SMB2 TREE_CONNECT Response	46
2.2.11 SMB2 TREE_DISCONNECT Request	49
2.2.12 SMB2 TREE_DISCONNECT Response	50
2.2.13 SMB2 CREATE Request	50
2.2.13.1 SMB2 Access Mask Encoding	55
2.2.13.1.1 File_Pipe_Printer_Access_Mask	55
2.2.13.1.2 Directory_Access_Mask	57
2.2.13.2 SMB2_CREATE_CONTEXT Request Values	58
2.2.13.2.1 SMB2_CREATE_EA_BUFFER	61
2.2.13.2.2 SMB2_CREATE_SD_BUFFER	61
2.2.13.2.3 SMB2_CREATE_DURABLE_HANDLE_REQUEST	61
2.2.13.2.4 SMB2_CREATE_DURABLE_HANDLE_RECONNECT	62
2.2.13.2.5 SMB2_CREATE_QUERY_MAXIMAL_ACCESS_REQUEST	62
2.2.13.2.6 SMB2_CREATE_ALLOCATION_SIZE	62
2.2.13.2.7 SMB2_CREATE_TIMEWARP_TOKEN	63
2.2.13.2.8 SMB2_CREATE_REQUESTLEASE	63
2.2.13.2.9 SMB2_CREATE_QUERY_ON_DISK_ID	64
2.2.13.2.10 SMB2_CREATE_REQUESTLEASE_V2	64
2.2.13.2.11 SMB2_CREATE_DURABLE_HANDLE_REQUEST_V2	65
2.2.13.2.12 SMB2_CREATE_DURABLE_HANDLE_RECONNECT_V2	66
2.2.13.2.13 SMB2_CREATE_APP_INSTANCE_ID	67

2.2.14 SMB2 CREATE Response.....	68
2.2.14.1 SMB2_FILEID	70
2.2.14.2 SMB2_CREATE_CONTEXT Response Values	71
2.2.14.2.1 SMB2_CREATE_EA_BUFFER	71
2.2.14.2.2 SMB2_CREATE_SD_BUFFER.....	71
2.2.14.2.3 SMB2_CREATE_DURABLE_HANDLE_RESPONSE	71
2.2.14.2.4 SMB2_CREATE_DURABLE_HANDLE_RECONNECT	72
2.2.14.2.5 SMB2_CREATE_QUERY_MAXIMAL_ACCESS_RESPONSE.....	72
2.2.14.2.6 SMB2_CREATE_APP_INSTANCE_ID.....	72
2.2.14.2.7 SMB2_CREATE_ALLOCATION_SIZE.....	72
2.2.14.2.8 SMB2_CREATE_TIMEWARP_TOKEN.....	72
2.2.14.2.9 SMB2_CREATE_QUERY_ON_DISK_ID.....	73
2.2.14.2.10 SMB2_CREATE_RESPONSE_LEASE.....	73
2.2.14.2.11 SMB2_CREATE_RESPONSELEASE_V2	74
2.2.14.2.12 SMB2_CREATE_DURABLE_HANDLE_RESPONSE_V2.....	76
2.2.14.2.13 SMB2_CREATE_DURABLE_HANDLE_RECONNECT_V2	76
2.2.15 SMB2 CLOSE Request	76
2.2.16 SMB2 CLOSE Response	77
2.2.17 SMB2 FLUSH Request.....	79
2.2.18 SMB2 FLUSH Response	80
2.2.19 SMB2 READ Request	80
2.2.20 SMB2 READ Response	82
2.2.21 SMB2 WRITE Request	83
2.2.22 SMB2 WRITE Response	84
2.2.23 SMB2 OPLOCK_BREAK Notification.....	85
2.2.23.1 Oplock Break Notification	85
2.2.23.2 Lease Break Notification	86
2.2.24 SMB2 OPLOCK_BREAK Acknowledgment.....	88
2.2.24.1 Oplock Break Acknowledgment.....	88
2.2.24.2 Lease Break Acknowledgment	89
2.2.25 SMB2 OPLOCK_BREAK Response	90
2.2.25.1 Oplock Break Response	90
2.2.25.2 Lease Break Response	91
2.2.26 SMB2 LOCK Request	92
2.2.26.1 SMB2_LOCK_ELEMENT Structure	93
2.2.27 SMB2 LOCK Response	94
2.2.28 SMB2 ECHO Request.....	94
2.2.29 SMB2 ECHO Response.....	95
2.2.30 SMB2 CANCEL Request.....	95
2.2.31 SMB2 IOCTL Request	95
2.2.31.1 SRV_COPYCHUNK_COPY	98
2.2.31.1.1 SRV_COPYCHUNK.....	99
2.2.31.2 SRV_READ_HASH Request	99
2.2.31.3 NETWORK_RESILIENCY_REQUEST Request.....	100
2.2.31.4 VALIDATE_NEGOTIATE_INFO Request.....	101
2.2.32 SMB2 IOCTL Response	102
2.2.32.1 SRV_COPYCHUNK_RESPONSE	103
2.2.32.2 SRV_SNAPSHOT_ARRAY	104
2.2.32.3 SRV_REQUEST_RESUME_KEY Response	104
2.2.32.4 SRV_READ_HASH Response	105
2.2.32.4.1 HASH_HEADER	105
2.2.32.4.2 SRV_HASH_RETRIEVE_HASH_BASED	107
2.2.32.4.3 SRV_HASH_RETRIEVE_FILE_BASED	107

2.2.32.5 NETWORK_INTERFACE_INFO Response	108
2.2.32.5.1 SOCKADDR_STORAGE	109
2.2.32.5.1.1 SOCKADDR_IN	110
2.2.32.5.1.2 SOCKADDR_IN6	111
2.2.32.6 VALIDATE_NEGOTIATE_INFO Response	111
2.2.33 SMB2 QUERY_DIRECTORY Request	112
2.2.34 SMB2 QUERY_DIRECTORY Response	114
2.2.35 SMB2 CHANGE_NOTIFY Request	114
2.2.36 SMB2 CHANGE_NOTIFY Response	116
2.2.36.1 FILE_NOTIFY_INFORMATION	117
2.2.37 SMB2 QUERY_INFO Request	118
2.2.37.1 SMB2_QUERY_QUOTA_INFO	122
2.2.38 SMB2 QUERY_INFO Response	123
2.2.39 SMB2 SET_INFO Request	124
2.2.40 SMB2 SET_INFO Response	126
2.2.41 SMB2 TRANSFORM_HEADER	127
3 Protocol Details	129
3.1 Common Details	129
3.1.1 Abstract Data Model	129
3.1.1.1 Global	129
3.1.2 Timers	129
3.1.3 Initialization	129
3.1.4 Higher-Layer Triggered Events	129
3.1.4.1 Signing An Outgoing Message	129
3.1.4.2 Generating Cryptographic Keys	130
3.1.5 Processing Events and Sequencing Rules	130
3.1.5.1 Verifying an Incoming Message	130
3.1.6 Timer Events	131
3.1.7 Other Local Events	131
3.2 Client Details	131
3.2.1 Abstract Data Model	131
3.2.1.1 Global	131
3.2.1.2 Per SMB2 Transport Connection	131
3.2.1.3 Per Session	133
3.2.1.4 Per Tree Connect	134
3.2.1.5 Per Open File	134
3.2.1.6 Per Application Open of a File	134
3.2.1.7 Per Pending Request	136
3.2.1.8 Per Channel	136
3.2.2 Timers	136
3.2.2.1 Request Expiration Timer	136
3.2.2.2 Idle Connection Timer	136
3.2.3 Initialization	136
3.2.4 Higher-Layer Triggered Events	137
3.2.4.1 Sending Any Outgoing Message	137
3.2.4.1.1 Signing the Message	137
3.2.4.1.2 Requesting Credits from the Server	138
3.2.4.1.3 Associating the Message with a MessageId	138
3.2.4.1.4 Sending Compounded Requests	138
3.2.4.1.5 Sending Multi-Credit Requests	139
3.2.4.1.6 Algorithm for Handling Available Message Sequence Numbers by the Client	139

3.2.4.1.7	Selecting a Channel	140
3.2.4.1.8	Encrypting the Message	140
3.2.4.2	Application Requests a Connection to a Share	141
3.2.4.2.1	Connecting to the Target Server	143
3.2.4.2.2	Negotiating the Protocol	144
3.2.4.2.2.1	Multi-Protocol Negotiate	144
3.2.4.2.2.2	SMB2-Only Negotiate	144
3.2.4.2.3	Authenticating the User	145
3.2.4.2.3.1	Application Requests Reauthenticating a User	146
3.2.4.2.4	Connecting to the Share.....	147
3.2.4.3	Application Requests Opening a File	148
3.2.4.3.1	Application Requests Opening a Named Pipe.....	150
3.2.4.3.2	Application Requests Sending a File to Print.....	150
3.2.4.3.3	Application Requests Creating a File with Extended Attributes.....	150
3.2.4.3.4	Application Requests Creating a File with a Security Descriptor	150
3.2.4.3.5	Application Requests Creating a File Opened for Durable Operation	150
3.2.4.3.6	Application Requests Opening a Previous Version of a File	151
3.2.4.3.7	Application Requests Creating a File with a Specific Allocation Size	151
3.2.4.3.8	Requesting a Lease on a File or a Directory	151
3.2.4.3.9	Application Requests Maximal Access Information of a File	152
3.2.4.3.10	Application Requests Identifier of a File	152
3.2.4.3.11	Application Supplies its Identifier.....	152
3.2.4.4	Re-establishing a Durable Open.....	152
3.2.4.5	Application Requests Closing a File or Named Pipe	153
3.2.4.6	Application Requests Reading from a File or Named Pipe	154
3.2.4.7	Application Requests Writing to a File or Named Pipe	155
3.2.4.8	Application Requests Querying File Attributes	157
3.2.4.9	Application Requests Applying File Attributes.....	158
3.2.4.10	Application Requests Querying File System Attributes.....	159
3.2.4.11	Application Requests Applying File System Attributes	160
3.2.4.12	Application Requests Querying File Security	161
3.2.4.13	Application Requests Applying File Security	162
3.2.4.14	Application Requests Querying Quota Information.....	163
3.2.4.15	Application Requests Applying Quota Information	164
3.2.4.16	Application Requests Flushing Cached Data	165
3.2.4.17	Application Requests Enumerating a Directory	166
3.2.4.17.1	Application Requests Continuing a Directory Enumeration.....	167
3.2.4.18	Application Requests Change Notifications for a Directory	167
3.2.4.19	Application Requests Locking of an Array of Byte Ranges	168
3.2.4.20	Application Requests an IO Control Code Operation.....	170
3.2.4.20.1	Application Requests Enumeration of Previous Versions.....	170
3.2.4.20.2	Application Requests a Server-Side Data Copy	171
3.2.4.20.2.1	Application Requests a Source File Key	172
3.2.4.20.2.2	Application Requests a Server Side Data Copy	172
3.2.4.20.3	Application Requests DFS Referral Information.....	174
3.2.4.20.4	Application Requests a Pipe Transaction	175
3.2.4.20.5	Application Requests a Peek at Pipe Data	176
3.2.4.20.6	Application Requests a Pass-Through Operation	177
3.2.4.20.7	Application Requests Content Information for a File	178
3.2.4.20.8	Application Requests Resiliency on an Open File	180
3.2.4.20.9	Application Requests Waiting for a Connection to a Pipe	181
3.2.4.20.10	Application Requests Querying Server's Network Interfaces	182
3.2.4.21	Application Requests Unlocking of an Array of Byte Ranges	183

3.2.4.22 Application Requests Closing a Share Connection	184
3.2.4.23 Application Requests Terminating an Authenticated Context	184
3.2.4.24 Application Requests Canceling an Operation.....	185
3.2.4.25 Application Requests the Session Key for an Authenticated Context	185
3.2.4.26 Application Requests Number of Opens on a Tree Connect	185
3.2.4.27 Application Notifies Offline Status of a Server	186
3.2.4.28 Application Notifies Online Status of a Server	186
3.2.4.29 Application Requests Moving to a Server Instance.....	186
3.2.5 Processing Events and Sequencing Rules.....	187
3.2.5.1 Receiving Any Message	187
3.2.5.1.1 Decrypting the Message	187
3.2.5.1.2 Finding the Application Request for This Response.....	187
3.2.5.1.3 Verifying the Signature	187
3.2.5.1.4 Granting Message Credits.....	188
3.2.5.1.5 Handling Asynchronous Responses	188
3.2.5.1.6 Handling Session Expiration.....	189
3.2.5.1.7 Handling Incorrectly Formatted Responses	189
3.2.5.1.8 Processing the Response	189
3.2.5.1.9 Handling Compounded Responses	189
3.2.5.2 Receiving an SMB2 NEGOTIATE Response.....	189
3.2.5.3 Receiving an SMB2 SESSION_SETUP Response	190
3.2.5.3.1 Handling a New Authentication	191
3.2.5.3.2 Handling a Reauthentication	193
3.2.5.3.3 Handling Session Binding	194
3.2.5.4 Receiving an SMB2 LOGOFF Response.....	196
3.2.5.5 Receiving an SMB2 TREE_CONNECT Response	196
3.2.5.6 Receiving an SMB2 TREE_DISCONNECT Response	199
3.2.5.7 Receiving an SMB2 CREATE Response for a New Create Operation.....	199
3.2.5.7.1 SMB2_CREATE_DURABLE_HANDLE_RESPONSE Create Context.....	200
3.2.5.7.2 SMB2_CREATE_QUERY_MAXIMAL_ACCESS_RESPONSE Create Context	200
3.2.5.7.3 SMB2_CREATE_QUERY_ON_DISK_ID Create Context	201
3.2.5.7.4 SMB2_CREATE_RESPONSELEASE Create Context	201
3.2.5.7.5 SMB2_CREATE_RESPONSELEASE_V2 Create Context	201
3.2.5.7.6 SMB2_CREATE_DURABLE_HANDLE_RESPONSE_V2 Create Context	202
3.2.5.8 Receiving an SMB2 CREATE Response for an Open Reestablishment	202
3.2.5.9 Receiving an SMB2 CLOSE Response.....	203
3.2.5.10 Receiving an SMB2 FLUSH Response	203
3.2.5.11 Receiving an SMB2 READ Response.....	203
3.2.5.12 Receiving an SMB2 WRITE Response	204
3.2.5.13 Receiving an SMB2 LOCK Response.....	204
3.2.5.14 Receiving an SMB2 IOCTL Response	204
3.2.5.14.1 Handling an Enumeration of Previous Versions Response.....	204
3.2.5.14.2 Handling a Server-Side Data Copy Source File Key Response	205
3.2.5.14.3 Handling a Server-Side Data Copy Response	205
3.2.5.14.4 Handling a DFS Referral Information Response.....	205
3.2.5.14.5 Handling a Pipe Transaction Response.....	205
3.2.5.14.6 Handling a Peek at Pipe Data Response	206
3.2.5.14.7 Handling a Content Information Retrieval Response.....	206
3.2.5.14.8 Handling a Pass-Through Operation Response.....	206
3.2.5.14.9 Handling a Resiliency Response.....	206
3.2.5.14.10 Handling a Pipe Wait Response.....	207
3.2.5.14.11 Handling a Network Interfaces Response	207
3.2.5.14.12 Handling a Validate Negotiate Info Response.....	207

3.2.5.15 Receiving an SMB2 QUERY_DIRECTORY Response	207
3.2.5.16 Receiving an SMB2 CHANGE_NOTIFY Response	207
3.2.5.17 Receiving an SMB2 QUERY_INFO Response	208
3.2.5.18 Receiving an SMB2 SET_INFO Response	208
3.2.5.19 Receiving an SMB2 OPLOCK_BREAK Notification	208
3.2.5.19.1 Receiving an Oplock Break Notification.....	208
3.2.5.19.2 Receiving a Lease Break Notification	209
3.2.5.19.3 Receiving an Oplock Break Acknowledgment Response	210
3.2.5.19.4 Receiving a Lease Break Acknowledgment Response.....	210
3.2.6 Timer Events	210
3.2.6.1 Request Expiration Timer Event.....	210
3.2.6.2 Idle Connection Timer Event	211
3.2.7 Other Local Events	211
3.2.7.1 Handling a Network Disconnect	211
3.3 Server Details	212
3.3.1 Abstract Data Model	212
3.3.1.1 Algorithm for Handling Available Message Sequence Numbers by the Server ...	213
3.3.1.2 Algorithm for the Granting of Credits.....	213
3.3.1.3 Algorithm for Change Notifications in an Object Store	214
3.3.1.4 Algorithm for Leasing in an Object Store.....	214
3.3.1.5 Global	215
3.3.1.6 Per Share	217
3.3.1.7 Per Transport Connection	218
3.3.1.8 Per Session	219
3.3.1.9 Per Tree Connect	220
3.3.1.10 Per Open	220
3.3.1.11 Per Lease Table	223
3.3.1.12 Per Lease.....	223
3.3.1.13 Per Request	224
3.3.1.14 Per Channel	224
3.3.2 Timers	224
3.3.2.1 Oplock Break Acknowledgment Timer.....	224
3.3.2.2 Durable Open Scavenger Timer	225
3.3.2.3 Session Expiration Timer	225
3.3.2.4 Resilient Open Scavenger Timer	225
3.3.3 Initialization	225
3.3.4 Higher-Layer Triggered Events	226
3.3.4.1 Sending Any Outgoing Message.....	226
3.3.4.1.1 Signing the Message.....	226
3.3.4.1.2 Granting Credits to the Client	227
3.3.4.1.3 Sending Compounded Responses	227
3.3.4.1.4 Encrypting the Message	227
3.3.4.2 Sending an Interim Response for an Asynchronous Operation	228
3.3.4.3 Sending a Success Response.....	229
3.3.4.4 Sending an Error Response	230
3.3.4.5 Server Application Requests Session Key of the Client.....	231
3.3.4.6 Object Store Indicates an Oplock Break.....	231
3.3.4.7 Object Store Indicates a Lease Break	232
3.3.4.8 DFS Server Notifies SMB2 Server That DFS Is Active.....	233
3.3.4.9 DFS Server Notifies SMB2 Server That a Share Is a DFS Share.....	233
3.3.4.10 DFS Server Notifies SMB2 Server That a Share Is Not a DFS Share	233
3.3.4.11 Server Application Requests Security Context of the Client.....	233
3.3.4.12 Server Application Requests Closing a Session.....	233

3.3.4.13	Server Application Registers a Share	234
3.3.4.14	Server Application Updates a Share.....	235
3.3.4.15	Server Application Dereisters a Share	236
3.3.4.16	Server Application Requests Querying a Share.....	236
3.3.4.17	Server Application Requests Closing an Open	237
3.3.4.18	Server Application Queries a Session.....	238
3.3.4.19	Server Application Queries a TreeConnect	238
3.3.4.20	Server Application Queries an Open	239
3.3.4.21	Server Application Requests Transport Binding Change.....	239
3.3.4.22	Server Application Enables the SMB2 Server	240
3.3.4.23	Server Application Disables the SMB2 Server	240
3.3.4.24	Server Application Requests Server Statistics	240
3.3.5	Processing Events and Sequencing Rules.....	241
3.3.5.1	Accepting an Incoming Connection.....	241
3.3.5.2	Receiving Any Message	242
3.3.5.2.1	Decrypting the Message	242
3.3.5.2.2	Verifying the Connection State.....	243
3.3.5.2.3	Verifying the Sequence Number	243
3.3.5.2.4	Verifying the Signature	243
3.3.5.2.5	Verifying the Credit Charge and the Payload Size	244
3.3.5.2.6	Handling Incorrectly Formatted Requests	244
3.3.5.2.7	Handling Compounded Requests	245
3.3.5.2.7.1	Handling Compounded Unrelated Requests	245
3.3.5.2.7.2	Handling Compounded Related Requests	245
3.3.5.2.8	Updating Idle Time.....	245
3.3.5.2.9	Verifying the Session	245
3.3.5.2.10	Verifying the Channel Sequence Number.....	246
3.3.5.2.11	Verifying the Tree Connect	246
3.3.5.3	Receiving an SMB_COM_NEGOTIATE.....	247
3.3.5.3.1	SMB 2.1 or SMB 3.0 Support	247
3.3.5.3.2	SMB 2.002 Support	248
3.3.5.4	Receiving an SMB2 NEGOTIATE Request.....	249
3.3.5.5	Receiving an SMB2 SESSION_SETUP Request	251
3.3.5.5.1	Authenticating a New Session	253
3.3.5.5.2	Reauthenticating an Existing Session	253
3.3.5.5.3	Handling GSS-API Authentication	253
3.3.5.6	Receiving an SMB2 LOGOFF Request.....	258
3.3.5.7	Receiving an SMB2 TREE_CONNECT Request	258
3.3.5.8	Receiving an SMB2 TREE_DISCONNECT Request	261
3.3.5.9	Receiving an SMB2 CREATE Request	262
3.3.5.9.1	Handling the SMB2_CREATE_EA_BUFFER Create Context	268
3.3.5.9.2	Handling the SMB2_CREATE_SD_BUFFER Create Context.....	268
3.3.5.9.3	Handling the SMB2_CREATE_ALLOCATION_SIZE Create Context.....	268
3.3.5.9.4	Handling the SMB2_CREATE_TIMEWARP_TOKEN Create Context.....	268
3.3.5.9.5	Handling the SMB2_CREATE_QUERY_MAXIMAL_ACCESS_REQUEST Create Context.....	269
3.3.5.9.6	Handling the SMB2_CREATE_DURABLE_HANDLE_REQUEST Create Context.....	269
3.3.5.9.7	Handling the SMB2_CREATE_DURABLE_HANDLE_RECONNECT Create Context.....	270
3.3.5.9.8	Handling the SMB2_CREATE_REQUESTLEASE Create Context	271
3.3.5.9.9	Handling the SMB2_CREATE_QUERY_ON_DISK_ID Create Context.....	273

3.3.5.9.10	Handling the SMB2_CREATE_DURABLE_HANDLE_REQUEST_V2 Create Context.....	273
3.3.5.9.11	Handling the SMB2_CREATE_REQUESTLEASE_V2 Create Context	275
3.3.5.9.12	Handling the SMB2_CREATE_DURABLE_HANDLE_RECONNECT_V2 Create Context.....	277
3.3.5.9.13	Handling the SMB2_CREATE_APP_INSTANCE_ID Create Context	279
3.3.5.10	Receiving an SMB2 CLOSE Request	279
3.3.5.11	Receiving an SMB2 FLUSH Request	280
3.3.5.12	Receiving an SMB2 READ Request.....	281
3.3.5.13	Receiving an SMB2 WRITE Request	283
3.3.5.14	Receiving an SMB2 LOCK Request.....	285
3.3.5.14.1	Processing Unlocks	286
3.3.5.14.2	Processing Locks	287
3.3.5.15	Receiving an SMB2 IOCTL Request.....	288
3.3.5.15.1	Handling an Enumeration of Previous Versions Request.....	290
3.3.5.15.2	Handling a DFS Referral Information Request.....	291
3.3.5.15.3	Handling a Pipe Transaction Request.....	291
3.3.5.15.4	Handling a Peek at Pipe Data Request	292
3.3.5.15.5	Handling a Source File Key Request	293
3.3.5.15.6	Handling a Server-Side Data Copy Request.....	294
3.3.5.15.6.1	Sending a Copy Failure Server-Side Copy Response	295
3.3.5.15.6.2	Sending an Invalid Parameter Server-Side Copy Response.....	295
3.3.5.15.7	Handling a Content Information Retrieval Request.....	296
3.3.5.15.8	Handling a Pass-Through Operation Request.....	298
3.3.5.15.9	Handling a Resiliency Request.....	299
3.3.5.15.10	Handling a Pipe Wait Request.....	300
3.3.5.15.11	Handling a Query Network Interface Request	300
3.3.5.15.12	Handling a Validate Negotiate Info Request.....	301
3.3.5.15.13	Handling a Set Reparse Point Request	302
3.3.5.15.14	Handling a File Level Trim Request	302
3.3.5.16	Receiving an SMB2 CANCEL Request	302
3.3.5.17	Receiving an SMB2 ECHO Request	303
3.3.5.18	Receiving an SMB2 QUERY_DIRECTORY Request	303
3.3.5.19	Receiving an SMB2 CHANGE_NOTIFY Request	306
3.3.5.20	Receiving an SMB2 QUERY_INFO Request	307
3.3.5.20.1	Handling SMB2_0_INFO_FILE	309
3.3.5.20.2	Handling SMB2_0_INFO_FILESYSTEM	310
3.3.5.20.3	Handling SMB2_0_INFO_SECURITY	311
3.3.5.20.4	Handling SMB2_0_INFO_QUOTA	311
3.3.5.21	Receiving an SMB2_SET_INFO Request	313
3.3.5.21.1	Handling SMB2_0_INFO_FILE	314
3.3.5.21.2	Handling SMB2_0_INFO_FILESYSTEM	315
3.3.5.21.3	Handling SMB2_0_INFO_SECURITY	315
3.3.5.21.4	Handling SMB2_0_INFO_QUOTA	315
3.3.5.22	Receiving an SMB2_OPLOCK_BREAK Acknowledgment	316
3.3.5.22.1	Processing an Oplock Acknowledgment	316
3.3.5.22.2	Processing a Lease Acknowledgment.....	317
3.3.6	Timer Events	318
3.3.6.1	Oplock Break Acknowledgment Timer Event.....	318
3.3.6.2	Durable Open Scavenger Timer Event	318
3.3.6.3	Session Expiration Timer Event	319
3.3.6.4	Resilient Open Scavenger Timer Event	319
3.3.7	Other Local Events	319

3.3.7.1 Handling Loss of a Connection.....	319
4 Protocol Examples.....	321
4.1 Connecting to a Share by Using a Multi-Protocol Negotiate.....	321
4.2 Negotiating SMB 2.10 dialect by using Multi-Protocol Negotiate	327
4.3 Connecting to a Share by Using an SMB2 Negotiate	332
4.4 Executing an Operation on a Named Pipe.....	338
4.5 Reading from a Remote File	347
4.6 Writing to a Remote File	353
4.7 Disconnecting a Share and Logging Off.....	363
4.8 Establish Alternate Channel.....	365
4.9 Replay Create Request on an Alternate Channel	376
5 Security.....	381
5.1 Security Considerations for Implementers.....	381
5.2 Index of Security Parameters	381
6 Appendix A: Product Behavior.....	382
7 Change Tracking.....	410
8 Index	418

1 Introduction

The Server Message Block (SMB) Protocol Versions 2 and 3 supports the sharing of file and print resources between machines. The protocol borrows and extends concepts from the Server Message Block (SMB) Version 1.0 Protocol, as specified in [\[MS-SMB\]](#). This specification assumes familiarity with [\[MS-SMB\]](#), and with the security concepts described in [\[MS-WPO\]](#) section 8.

Sections 1.8, 2, and 3 of this specification are normative and can contain the terms MAY, SHOULD, MUST, MUST NOT, and SHOULD NOT as defined in RFC 2119. Sections 1.5 and 1.9 are also normative but cannot contain those terms. All other sections and examples in this specification are informative.

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
IPv4
IPv6
main stream
named pipe
named stream
NetBIOS
network byte order
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 [\[MS-PCCRR\]](#).

channel: A logical entity that associates a transport connection to a session.

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 [2.2.32.4.1](#)).

create context: A variable-length attribute that is sent with an [SMB2 CREATE Request \(section 2.2.13\)](#) or [SMB2 CREATE Response \(section 2.2.14\)](#) that either gives extra information about how the create will 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 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.

Olock: 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.

Olock 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 Specifications documentation 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] FIPS PUBS, "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](#)".

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

[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](#)".

[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](#)".

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

- [MS-SRVS] Microsoft Corporation, "[Server Service Remote Protocol](#)".
- [SP800-108] National Institute of Standards and Technology. "Special Publication 800-108, Recommendation for Key Derivation Using Pseudorandom Functions", October 2009, <http://csrc.nist.gov/publications/nistpubs/800-108/sp800-108.pdf>
- [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>
- [RFC4309] Housley, R., "Using Advanced Encryption Standard (AES) CCM Mode with IPsec Encapsulating Security Payload (ESP)", RFC 4309, December 2005, <http://www.ietf.org/rfc/rfc4309.txt>
- [RFC4493] Song, JH., Poovendran, R., Lee, J., and Iwata, T., "The AES-CMAC Algorithm", RFC 4493, June 2006, <http://www.ietf.org/rfc/rfc4493.txt>
- [RFC5084] Housley, R., "Using AES-CCM and AES-GCM Authenticated Encryption in the Cryptographic Message Syntax (CMS)", RFC 5084, November 2007, <http://www.ietf.org/rfc/rfc5084.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](#)".
- [MS-PCCRR] Microsoft Corporation, "[Peer Content Caching and Retrieval: Retrieval Protocol](#)".
- [MS-SMBD] Microsoft Corporation, "[SMB2 Remote Direct Memory Access \(RDMA\) Transport Protocol](#)".
- [MS-SWN] Microsoft Corporation, "[Service Witness Protocol](#)".
- [MS-WPO] Microsoft Corporation, "[Windows Protocols 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>

[MSDN-SOCKADDR_STORAGE] Microsoft Corporation, "SOCKADDR_STORAGE structure", [http://msdn.microsoft.com/en-us/library/ms740504\(VS.85\).aspx](http://msdn.microsoft.com/en-us/library/ms740504(VS.85).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) Protocol Versions 2 and 3, hereafter referred to as "SMB 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 [\[MS-PCCRC\]](#) section [2.3](#).

The SMB 3.0 dialect introduces the following enhancements:

- Allowing a client to retrieve hashes for a particular region of a file for use in branch cache retrieval, as specified in [\[MS-PCCRC\]](#) section 2.4.
- Allowing a client to obtain lease on a directory.
- Supporting the encryption of traffic between client and server on a per-share basis.
- Supporting the use of Remote Direct Memory Access (RDMA) transports, when the appropriate hardware and network are available.
- Supporting enhanced failover between client and server, including optional handle persistence.
- Allowing a client to bind a session to multiple connections to the server. A request can be sent through any channel associated to the session, and the corresponding response is sent through the same channel as used by the request. The following diagram shows an example of two sessions using multiple channels to the server.

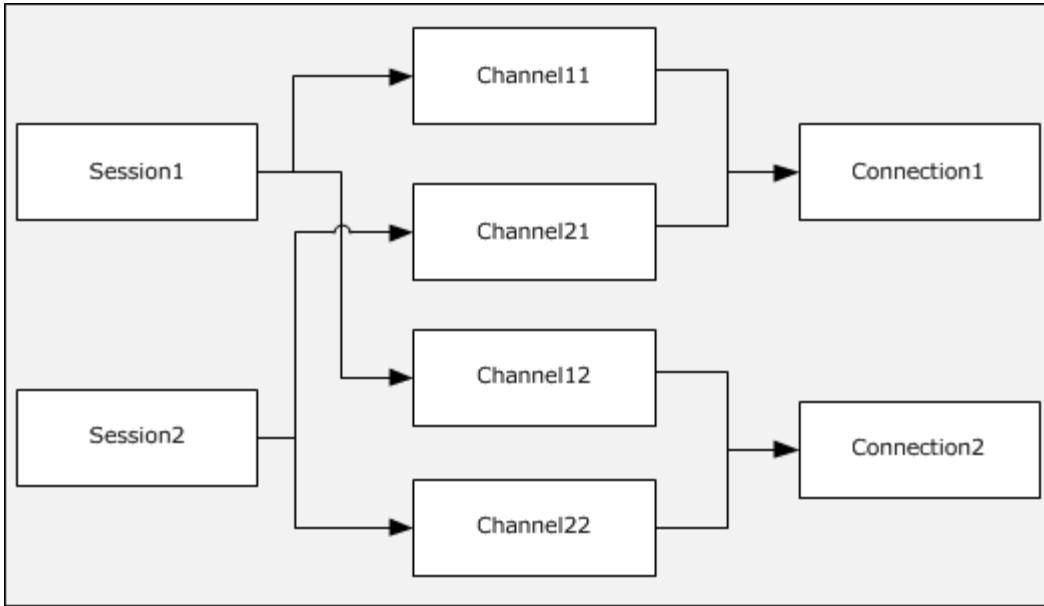


Figure 1: Two sessions using multiple channels

1.4 Relationship to Other Protocols

The SMB 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. The SMB 3.0 dialect also supports the use of RDMA as a transport.

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 and 3.0 dialect.

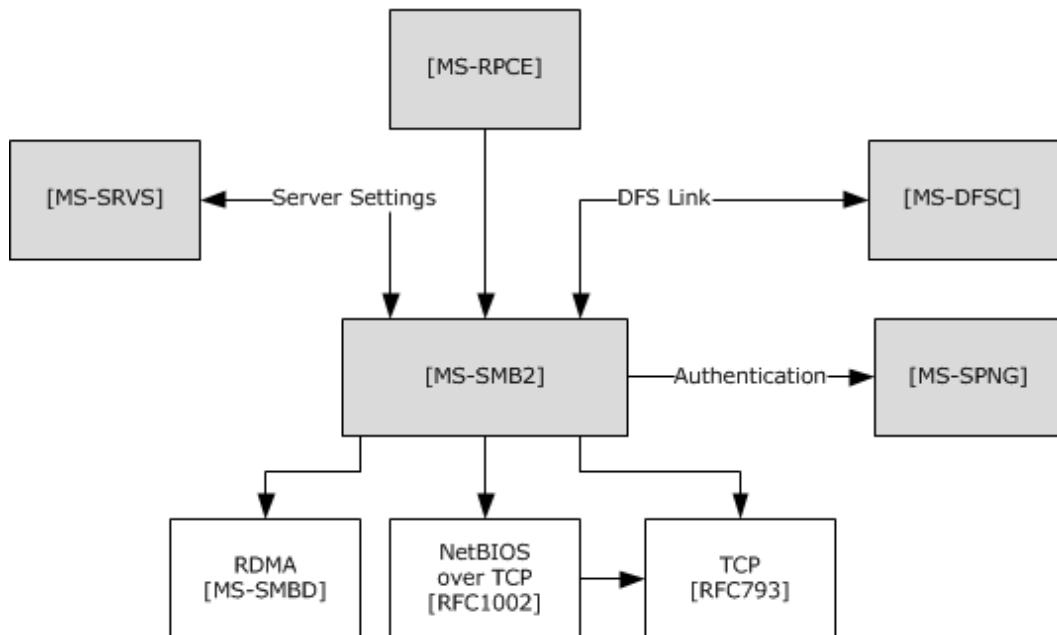


Figure 2: Relationship to other protocols

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, NetBIOS over TCP, or RDMA as underlying transports.

1.5 Prerequisites/Preconditions

The SMB 2 Protocol assumes the availability of the following resources:

- Either TCP or NetBIOS over TCP to support reliable, in-order message delivery. The SMB 3.0 dialect optionally supports the additional use of RDMA to support reliable, in-order message delivery with direct placement.

- 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 SMB 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.
- Encryption of client/server traffic when the SMB 3.0 dialect is negotiated.

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, TCP, or RDMA, 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 [\[MS-SPNG\]](#). When the SMB 3.0 dialect is negotiated, encryption is supported on a per-share basis through the use of AES_CMAC-128, as specified in [\[RFC4493\]](#).
- 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](#)).
 - 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 three dialects of the SMB 2 Protocol:

- Negotiating the SMB 3.0 dialect implies support for the requests and responses as specified in this document, except those explicitly marked for the SMB 2.002 or SMB 2.1 dialect.
- Negotiating the SMB 2.1 dialect implies support for the requests and responses as specified in this document, except those explicitly marked for the SMB 2.002 or SMB 3.0 dialects.
- 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 or 3.0 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 on the server implementing the SMB 2 Protocol dialects. 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.

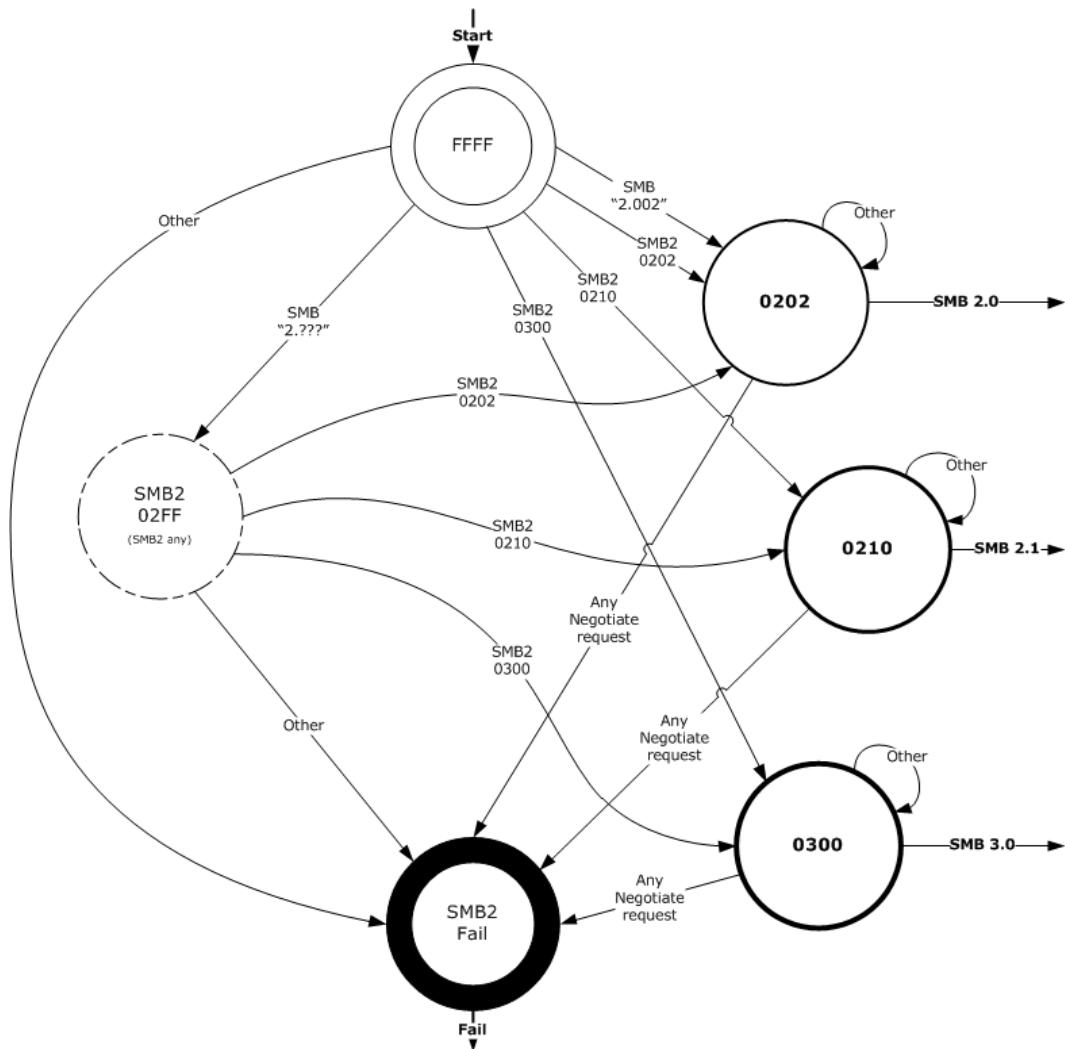


Figure 3: Connection.NegotiateDialect state transitions in SMB 2 Protocol server

1.8 Vendor-Extensible Fields

There are no vendor-extensible fields for the SMB 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 [\[RFC1002\]](#). When the SMB 3.0 dialect is negotiated and an RDMA transport is used, the standards assignment for the protocol specified in [\[MS-SMBD\]](#) is used.

2 Messages

The following sections specify how SMB 2 Protocol messages are encapsulated on the wire and common SMB 2 Protocol data types.

2.1 Transport

The SMB 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).

The SMB 2 Protocol also supports SMB2 Remote Direct Memory Access [\[MS-SMBD\]](#) as transport.

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)

The SMB 3.0 dialect in the SMB 2 Protocol further enhances the following categories of messages in the SMB 2 Protocol:

- Protocol Negotiation and secure dialect validation (SMB2 NEGOTIATE, SMB2 IOCTL)

- Share Access (SMB2 TREE_CONNECT)
- File Access (SMB2 CREATE, SMB2 READ, SMB2 WRITE)
- Hash Retrieval (SMB2 IOCTL)
- Encryption (SMB2 TRANSFORM_HEADER)

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 Packet 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](#), [3.2.5.1.5](#), 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 Packet 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	1	0	1	2	3	4	5	6	7	8	9	2	0	1	2	3	4	5	6	7	8	9	3	0	1						
ProtocolId																																								
StructureSize																									CreditCharge															
(ChannelSequence/Reserved)/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 all other dialects, this field indicates the number of credits that this request consumes.

(ChannelSequence/Reserved)/Status (4 bytes): In a request, this field is interpreted in different ways depending on the SMB2 dialect.

In the SMB 3.0 dialect, this field is interpreted as the **ChannelSequence** field followed by the **Reserved** field in a request.

ChannelSequence (2 bytes): This field is an indication to the server about the client's **Channel** change.

Reserved (2 bytes): This field SHOULD be set to zero and the server MUST ignore it on receipt.

In the SMB 2.002 and SMB 2.1 dialects, this field is interpreted as the **Status** field in a request.

Status (4 bytes): The client MUST set this field to 0 and the server MUST ignore it on receipt.

In all SMB dialects for a response this field is interpreted as the **Status** field. 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

Name	Value
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_REDIRECTION 0x00000001	When set, indicates the message is a response rather than a request. This MUST be set on responses sent from the server to the client, and MUST NOT be set on requests sent from the client to the server.
SMB2_FLAGS_ASYNC_COMMAND 0x00000002	When set, indicates that this is an ASYNC SMB2 header. 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 3.3.5.2.7.2 .
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 .
SMB2_FLAGS_REPLAY_OPERATION 0x20000000	This flag is only valid for the SMB 3.0 dialect. When set, it indicates that this command is a replay operation. The client MUST ignore this bit on receipt.

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	1	0	1	2	3	4	5	6	7	8	9	2	0	1	2	3	4	5	6	7	8	9	3	0	1
ProtocolId																																		
StructureSize																CreditCharge																		
(ChannelSequence/Reserved)/Status																																		
Command																CreditRequest/CreditResponse																		
Flags																																		
NextCommand																																		
MessageId																																		
...																																		
Reserved																																		
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 all other dialects, this field indicates the number of credits that this request consumes.

(ChannelSequence/Reserved)/Status (4 bytes): In a request, this field is interpreted in different ways depending on the SMB2 dialect.

In the SMB 3.0 dialect, this field is interpreted as the **ChannelSequence** field followed by the **Reserved** field in a request.

ChannelSequence (2 bytes): This field is an indication to the server about the client's **Channel** change.

Reserved (2 bytes): This field SHOULD be set to zero and the server MUST ignore it on receipt.

In the SMB 2.002 and SMB 2.1 dialects, this field is interpreted as the **Status** field in a request.

Status (4 bytes): The client MUST set this field to 0 and the server MUST ignore it on receipt.

In all SMB dialects for a response this field is interpreted as the **Status** field. This field can be set to any value. For a list of valid status codes, see [\[MS-ERREF\]](#) section 2.3.

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

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

Name	Value
SMB2_TREE_CONNECT	0x0003
SMB2_TREE_DISCONNECT	0x0004
SMB2_CREATE	0x0005
SMB2_CLOSE	0x0006
SMB2_FLUSH	0x0007
SMB2_READ	0x0008
SMB2_WRITE	0x0009
SMB2_LOCK	0x000A
SMB2_IOCTL	0x000B
SMB2_CANCEL	0x000C
SMB2_ECHO	0x000D
SMB2_QUERY_DIRECTORY	0x000E
SMB2_CHANGE_NOTIFY	0x000F
SMB2_QUERY_INFO	0x0010
SMB2_SET_INFO	0x0011
SMB2_OPLOCK_BREAK	0x0012

CreditRequest/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_REDIRECTION 0x00000001	When set, indicates the message is a response, rather than a request. This MUST be set on responses sent from the server to the client and MUST NOT be set on requests sent from the client to the server.
SMB2_FLAGS_ASYNC_COMMAND 0x00000002	When set, indicates that this is an ASYNC SMB2 header. 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

Value	Meaning
	operation in a compounded request chain. The use of this flag in an SMB2 response is as specified in 3.3.5.2.7.2 .
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 .
SMB2_FLAGS_REPLAY_OPERATION 0x20000000	This flag is only valid for the SMB 3.0 dialect. When set, it indicates that this command is a replay operation. The client MUST ignore this bit on receipt.

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.

Reserved (4 bytes): This field MUST NOT be used and MUST be reserved. The client SHOULD set this field to 0. The server 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** can 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	1	0	1	2	3	4	5	6	7	8	9	2	0	1	2	3	4	5	6	7	8	9	3	0	1													
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	1	0	1	2	3	4	5	6	7	8	9	2	0	1	2	3	4	5	6	7	8	9	3	0	1						
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	1	0	1	2	3	4	5	6	7	8	9	2	0	1	2	3	4	5	6	7	8	9	3	0	1
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 server MUST ignore this bit.
SMB2_NEGOTIATE_SIGNING_REQUIRED	When set, indicates that security signatures are

Value	Meaning
0x0002	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): If the client implements the SMB 3.0 dialect, the **Capabilities** field MUST be constructed using the following values. Otherwise, this field MUST be set to 0 and the server MUST ignore it on receipt.

Value	Meaning
SMB2_GLOBAL_CAP_DFS 0x00000001	When set, indicates that the client supports the Distributed File System (DFS).
SMB2_GLOBAL_CAP_LEASING 0x00000002	When set, indicates that the client supports leasing.
SMB2_GLOBAL_CAP_LARGE_MTU 0x00000004	When set, indicates that the client supports multi-credit operations.
SMB2_GLOBAL_CAP_MULTI_CHANNEL 0x00000008	When set, indicates that the client supports establishing multiple channels for a single session.
SMB2_GLOBAL_CAP_PERSISTENT_HANDLES 0x00000010	When set, indicates that the client supports persistent handles.
SMB2_GLOBAL_CAP_DIRECTORY_LEASING 0x00000020	When set, indicates that the client supports directory leasing.
SMB2_GLOBAL_CAP_ENCRYPTION 0x00000040	When set, indicates that the client supports encryption.

ClientGuid (16 bytes): It MUST be a **GUID** (as specified in [\[MS-DTYP\]](#) section 2.3.2.3) generated by the client, if the **Dialects** field contains a value other than 0x0202. 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. <u><8></u>
0x0210	SMB 2.1 dialect revision number. <u><9></u>
0x0300	SMB 3.0 dialect revision number. <u><10></u>

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	1	0	1	2	3	4	5	6	7	8	9	2	0	1	2	3	4	5	6	7	8	9	3	0	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.[<11>](#)

Value	Meaning
0x0202	SMB 2.002 dialect revision number. <u><12></u>
0x0210	SMB 2.1 dialect revision number. <u><13></u>
0x0300	SMB 3.0 dialect revision number. <u><14></u>
0x02FF	SMB2 wildcard revision number; indicates that the server implements 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. <u><15></u>

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.[<16>](#)

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.[<17>](#)

Capabilities (4 bytes): The Capabilities field specifies protocol capabilities for the server. This field MUST be constructed using a combination of zero or more of 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 is not valid for the SMB 2.002 dialect.
SMB2_GLOBAL_CAP_LARGE_MTU 0x00000004	When set, indicates that the server supports multi-credit operations. This flag is not valid for the SMB 2.002 dialect.

Value	Meaning
SMB2_GLOBAL_CAP_MULTI_CHANNEL 0x00000008	When set, indicates that the server supports establishing multiple channels for a single session. This flag is only valid for the SMB 3.0 dialect.
SMB2_GLOBAL_CAP_PERSISTENT_HANDLES 0x00000010	When set, indicates that the server supports persistent handles. This flag is only valid for the SMB 3.0 dialect.
SMB2_GLOBAL_CAP_DIRECTORY_LEASING 0x00000020	When set, indicates that the server supports directory leasing. This flag is only valid for the SMB 3.0 dialect.
SMB2_GLOBAL_CAP_ENCRYPTION 0x00000040	When set, indicates that the server supports encryption. This flag is only valid for the SMB 3.0 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.<18>

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	1	0	1	2	3	4	5	6	7	8	9	2	0	1	2	3	4	5	6	7	8	9	3	0	1													
StructureSize										Flags										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.

Flags (1 byte): If the client implements the SMB 3.0 dialect, this field MUST be set to combination of zero or more of the following values. Otherwise it MUST be set to 0.

Value	Meaning
SMB2_SESSION_FLAG_BINDING 0x01	When set, indicates that the request is to bind an existing session to a new connection.

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 0x02	When set, indicates that security signatures are required by the client.

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

Value	Meaning
SMB2_GLOBAL_CAP_DFS 0x00000001	When set, indicates that the client supports the Distributed File System (DFS).
SMB2_GLOBAL_CAP_UNUSED1 0x00000002	SHOULD be set to zero, and server MUST ignore.
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<19> 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 remove any session associated with this identifier. 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<20> 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	1	0	1	2	3	4	5	6	7	8	9	2	0	1	2	3	4	5	6	7	8	9	3	0	1
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.
SMB2_SESSION_FLAG_ENCRYPT_DATA 0x0004	If set, the server supports the encryption of messages in this session. This flag is only valid for the SMB 3.0 dialect.

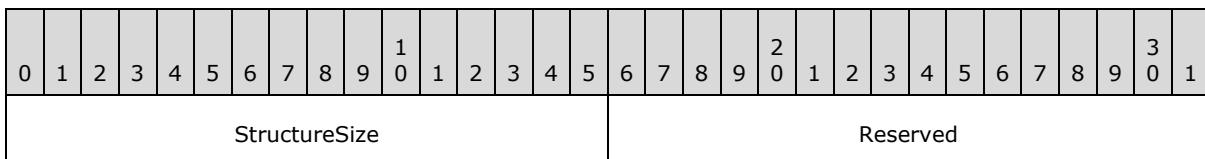
SecurityBufferOffset (2 bytes): The offset, in bytes, from the beginning of the SMB2 header to the security buffer.

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 [<21>](#) 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.



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	1	0	1	2	3	4	5	6	7	8	9	2	0	1	2	3	4	5	6	7	8	9	3	0	1
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	1	0	1	2	3	4	5	6	7	8	9	2	0	1	2	3	4	5	6	7	8	9	3	0	1													
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.<22>

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	1	0	1	2	3	4	5	6	7	8	9	2	0	1	2	3	4	5	6	7	8	9	3	0	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: SMB2_SHAREFLAG_DFS,
SMB2_SHAREFLAG_DFS_ROOT, SMB2_SHAREFLAG_RESTRICT_EXCLUSIVE_OPEN,
SMB2_SHAREFLAG_FORCE_SHARED_DELETE,
SMB2_SHAREFLAG_ALLOW_NAMESPACE_CACHING,
SMB2_SHAREFLAG_ACCESS_BASED_DIRECTORY_ENUM,
SMB2_SHAREFLAG_FORCE_LEVELII_OPLOCK and SMB2_SHAREFLAG_ENABLE_HASH.

Descriptions of the individual flags follow.

Value	Meaning
SMB2_SHAREFLAG_MANUAL_CACHING	The client may cache files that are

Value	Meaning
0x00000000	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.
SMB2_SHAREFLAG_DFS 0x00000001	The specified share is present in a Distributed File System (DFS) tree structure. The server SHOULD set the SMB2_SHAREFLAG_DFS bit in the ShareFlags field if the per-share property Share.IsDfs is TRUE.
SMB2_SHAREFLAG_DFS_ROOT 0x00000002	The specified share is present in a DFS tree structure. The server SHOULD set the SMB2_SHAREFLAG_DFS_ROOT bit in the ShareFlags field if the per-share property Share.IsDfs is TRUE.
SMB2_SHAREFLAG_RESTRICT_EXCLUSIVE_OPENS 0x00000100	The specified share disallows exclusive file opens that deny reads to an open file.
SMB2_SHAREFLAG_FORCE_SHARED_DELETE 0x00000200	Shared files in the specified share can be forcibly deleted.
SMB2_SHAREFLAG_ALLOW_NAMESPACE_CACHING 0x00000400	The client MUST ignore this flag.
SMB2_SHAREFLAG_ACCESS_BASED_DIRECTORY_ENUM 0x00000800	The server will filter directory entries based on the access permissions of the client.
SMB2_SHAREFLAG_FORCE_LEVELII_OPLOCK 0x00001000	The server will not issue exclusive caching rights on this share. <23>
SMB2_SHAREFLAG_ENABLE_HASH_V1 0x00002000	The share supports hash generation for branch cache retrieval of data. For more information, see section 2.2.31.2 . This value is only supported for the SMB 2.1 and 3.0 dialects.
SMB2_SHAREFLAG_ENABLE_HASH_V2 0x00004000	The share supports v2 hash generation for branch cache retrieval of data. For more information, see section 2.2.31.2 . This value is only supported for the SMB 3.0 dialect.

Value	Meaning
SMB2_SHAREFLAG_ENCRYPT_DATA 0x00008000	If set, the server supports the encryption of remote file access messages on this share. This flag is only valid for the SMB 3.0 dialect.

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

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-share property Share.IsDfs is TRUE.
SMB2_SHARE_CAP_CONTINUOUS_AVAILABILITY 0x00000010	The specified share is continuously available. This flag is only valid for the SMB 3.0 dialect.
SMB2_SHARE_CAP_SCALEOUT 0x00000020	The specified share is present on a server configuration which facilitates faster recovery of durable handles. This flag is only valid for the SMB 3.0 dialect.
SMB2_SHARE_CAP_CLUSTER 0x00000040	The specified share is present on a server configuration which provides monitoring of the availability of share through the Witness service specified in [MS-SWN] . This flag is only valid for the SMB 3.0 dialect.

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	1	0	1	2	3	4	5	6	7	8	9	2	0	1	2	3	4	5	6	7	8	9	3	0	1
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	1	0	1	2	3	4	5	6	7	8	9	2	0	1	2	3	4	5	6	7	8	9	3	0	1
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.

0	1	2	3	4	5	6	7	8	9	1	0	1	2	3	4	5	6	7	8	9	2	0	1	2	3	4	5	6	7	8	9	3	0	1
StructureSize																SecurityFlags																		
ImpersonationLevel																																		
SmbCreateFlags																																		
...																																		
Reserved																																		
...																																		
DesiredAccess																																		
FileAttributes																																		
ShareAccess																																		
CreateDisposition																																		
CreateOptions																																		

NameOffset	NameLength
	CreateContextsOffset
	CreateContextsLength
	Buffer (variable)
	...

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

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

RequestedOplockLevel (1 byte): The requested **oplock** level. This field MUST contain one of the following values. [<24>](#) For named pipes, the server MUST always revert to **SMB2_OPLOCK_LEVEL_NONE** irrespective of the value of this field.

Value	Meaning
SMB2_OPLOCK_LEVEL_NONE 0x00	No oplock is requested.
SMB2_OPLOCK_LEVEL_II 0x01	A level II oplock is requested.
SMB2_OPLOCK_LEVEL_EXCLUSIVE 0x08	An exclusive oplock is requested.
SMB2_OPLOCK_LEVEL_BATCH 0x09	A batch oplock is requested.
SMB2_OPLOCK_LEVEL_LEASE 0xFF	A lease is requested. If set, the request packet MUST contain an SMB2_CREATE_REQUESTLEASE (section 2.2.13.2.8) create context. This value is not valid for the SMB 2.002 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	The application-requested impersonation level is Impersonation.

Value	Meaning
0x00000002	
Delegate 0x00000003	The application-requested impersonation level is Delegate.

Impersonation is specified in [\[MS-WPO\]](#) section 8.4; for more information about impersonation, see [\[MSDN-IMPERS\]](#).

SmbCreateFlags (8 bytes): This field MUST NOT be used and MUST be reserved. The client SHOULD set this field to zero, 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.

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<25> 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.<26>
FILE_OPEN 0x00000001	If the file already exists, return success; otherwise, fail the operation. MUST NOT be used for a printer object.

Value	Meaning
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. <27>
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. <28>

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.[<29>](#)

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. <30>
FILE_SYNCHRONOUS_IO_NONALERT 0x00000020	This bit SHOULD be set to 0 and MUST be ignored by the server. <31>

Value	Meaning
FILE_NON_DIRECTORY_FILE 0x00000040	If the name of the file being created or opened matches with an existing directory file, 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. <u><32></u>
FILE_NO_EA KNOWLEDGE 0x00000200	The caller does not understand how to handle extended attributes. If the request includes an <u>SMB2_CREATE_EA_BUFFER</u> create context, then the server MUST fail this request with STATUS_ACCESS_DENIED. If extended attributes with the FILE_NEED_EA flag (see <u>[MS-FSCC]</u> section 2.4.15) set 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. <u><33></u>
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 can 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 can 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. <u><34></u>
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.

Value	Meaning
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. <35>

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	1	0	1	2	3	4	5	6	7	8	9	2	0	1	2	3	4	5	6	7	8	9	3	0	1
File_Pipe_Printer_Access_Mask																																		

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

Value	Meaning
FILE_READ_DATA 0x00000001	This value indicates the right to read data from the file or named pipe.
FILE_WRITE_DATA 0x00000002	This value indicates the right to write data into the file or named pipe beyond the end of the file.
FILE_APPEND_DATA 0x00000004	This value indicates the right to append data into the file or named pipe.
FILE_READ_EA 0x00000008	This value indicates the right to read the extended attributes of the file or named pipe.
FILE_WRITE_EA 0x00000010	This value indicates the right to write or change the extended attributes to the file or named pipe.
FILE_EXECUTE 0x00000020	This value indicates the right to execute the file.
FILE_READ_ATTRIBUTES 0x00000080	This value indicates the right to read the attributes of the file.
FILE_WRITE_ATTRIBUTES 0x00000100	This value indicates the right to change the attributes of the file.
DELETE 0x00010000	This value indicates the right to delete the file.
READ_CONTROL 0x00020000	This value indicates the right to read the security descriptor for the file or named pipe.
WRITE_DAC 0x00040000	This value indicates the right to change the discretionary access control list (DACL) in the security descriptor for the file or named pipe. For the DACL data structure, see ACL in [MS-DTYP].
WRITE_OWNER 0x00080000	This value indicates the right to change the owner in the security descriptor for the file or named pipe.
SYNCHRONIZE 0x00100000	SMB2 clients set this flag to any value. <36> SMB2 servers SHOULD <37> 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]. <38>
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.

Value	Meaning
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	1	0	1	2	3	4	5	6	7	8	9	2	0	1	2	3	4	5	6	7	8	9	3	0	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.

Value	Meaning
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. <39> SMB2 servers SHOULD <40> 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] . <41>
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	1	0	1	2	3	4	5	6	7	8	9	2	0	1	2	3	4	5	6	7	8	9	3	0	1													
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.

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.[<42>](#)

DataLength (4 bytes): The length, in bytes, of the data. The format of the data is determined by **NameOffset**, **NameLength**, **DataOffset**, and **DataLength**. The name is represented as four or more octets 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 and are defined to be in network byte order. More details are provided for each of these values in the following subsections.

Value	Meaning
SMB2_CREATE_EA_BUFFER 0x45787441	("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")

Value	Meaning
0x53656344	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 0x44486e51	("DHnQ") The client is requesting the open to be durable (see section 3.3.5.9.6).
SMB2_CREATE_DURABLE_HANDLE_RECONNECT 0x44486e43	("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 0x416c5369	("AISi") The data contains the required allocation size of the newly created file.
SMB2_CREATE_QUERY_MAXIMAL_ACCESS_REQUEST 0x4d784163	("MxAc") The client is requesting that the server return maximal access information.
SMB2_CREATE_TIMEWARP_TOKEN 0x54577270	("TWrp") The client is requesting that the server open an earlier version of the file identified by the provided time stamp.
SMB2_CREATE_QUERY_ON_DISK_ID 0x51466964	("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 0x52714c73	("RqLs") The client is requesting that the server return a lease . This value is only supported for the SMB 2.1 and 3.0 dialects.
SMB2_CREATE_REQUESTLEASE_V2 0x52714c73	("RqLs") The client is requesting that the server return a lease for a file or a directory. This value is only supported for the SMB 3.0 dialect. This context value is the same as the SMB2_CREATE_REQUESTLEASE value; the server differentiates these requests based on the value of the DataLength field.
SMB2_CREATE_DURABLE_HANDLE_REQUEST_V2 0x44483251	("DH2Q") The client is requesting the open to be durable. This value is only supported for the SMB 3.0 dialect.

Value	Meaning
SMB2_CREATE_DURABLE_HANDLE_RECONNECT_V2 0x44483243	("DH2C") The client is requesting to reconnect to a durable open after being disconnected. This value is only supported for the SMB 3.0 dialect.
SMB2_CREATE_APP_INSTANCE_ID 0x45BCA66AEFA7F74A9008FA462E144D74	The client is supplying an identifier provided by an application instance while opening a file. This value is only supported for the SMB 3.0 dialect.

2.2.13.2.1 SMB2_CREATE_EA_BUFFER

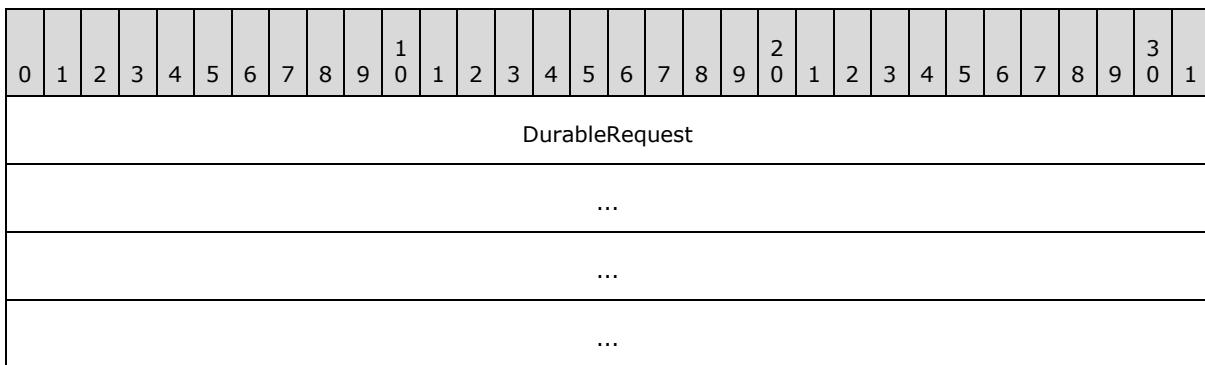
The SMB2_CREATE_EA_BUFFER context is specified on an [SMB2 CREATE Request](#) (section 2.2.13) when the client is applying extended attributes as part of creating a new file. The server MUST ignore this Create Context for requests to open an existing file, a pipe, or a printer. The extended attributes 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_REQUESTLEASE Create Context with a LeaseState that includes SMB2LEASEHANDLECACHING). 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.



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	1	0	1	2	3	4	5	6	7	8	9	2	0	1	2	3	4	5	6	7	8	9	3	0	1
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	1	0	1	2	3	4	5	6	7	8	9	2	0	1	2	3	4	5	6	7	8	9	3	0	1
Timestamp																																		
...																																		

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

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	1	0	1	2	3	4	5	6	7	8	9	2	0	1	2	3	4	5	6	7	8	9	3	0	1
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	1	0	1	2	3	4	5	6	7	8	9	2	0	1	2	3	4	5	6	7	8	9	3	0	1
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_REQUESTLEASE

The SMB2_CREATE_REQUESTLEASE 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 value is not valid for the SMB 2.002 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	1	0	1	2	3	4	5	6	7	8	9	2	0	1	2	3	4	5	6	7	8	9	3	0	1
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.[<43>](#)

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.

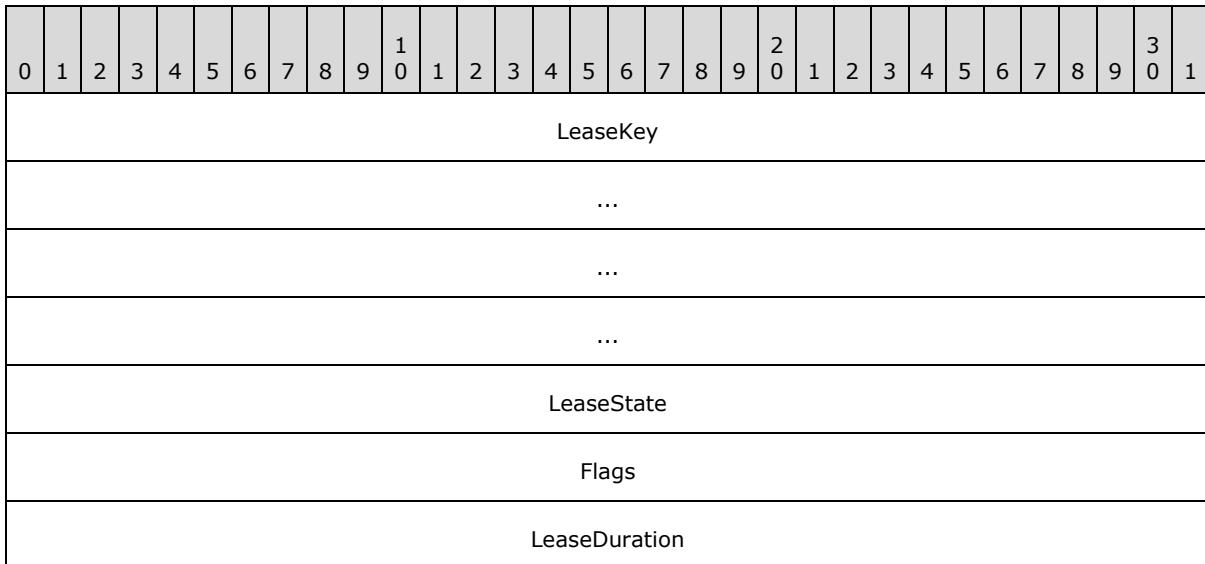
LeaseDuration (8 bytes): This field MUST NOT be used and MUST be reserved. The client MUST set this to 0, and the server MUST ignore it on receipt.

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.

2.2.13.2.10 SMB2_CREATE_REQUESTLEASE_V2

The SMB2_CREATE_REQUESTLEASE_V2 context is specified on an SMB2 CREATE Request when the client is requesting the server to return a lease on a file or a directory. This is only valid for the SMB 3.0 dialect. The data in the **Buffer** field of the SMB2_CREATE_CONTEXT (section [2.2.13.2](#)) structure MUST contain the following structure.



	...
	ParentLeaseKey
	...
	...
	...
Epoch	Reserved

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.[§44](#)

Value	Meaning
SMB2_LEASE_NONE 0x00000000	No lease is requested.
SMB2_LEASE_READ_CACHING 0x00000001	A read caching lease is requested.
SMB2_LEASE_HANDLE_CACHING 0x00000002	A handle caching lease is requested.
SMB2_LEASE_WRITE_CACHING 0x00000004	A write caching lease is requested.

Flags (4 bytes): This field MUST be set as a combination of the following values.

Value	Meaning
SMB2_LEASE_FLAG_PARENT_LEASE_KEY_SET 0x00000004	When set, indicates that the ParentLeaseKey is set.

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.

ParentLeaseKey (16 bytes): A unique key that identifies the owner of the lease for the parent directory.

Epoch (2 bytes): A 16-bit unsigned integer used to track lease state changes.

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.

2.2.13.2.11 SMB2_CREATE_DURABLE_HANDLE_REQUEST_V2

The SMB2_CREATE_DURABLE_HANDLE_REQUEST_V2 context is only valid for the SMB 3.0 dialect. When the client is not requesting a persistent handle, the client SHOULD also request a batch oplock

(by setting **RequestedOlockLevel** to SMB2_OPLOCK_LEVEL_BATCH) or a handle caching lease (by using an SMB2_CREATE_REQUESTLEASE or SMB2_CREATE_REQUESTLEASE_V2 Create Context with a **LeaseState** that includes SMB2_LEASE_HANDLE_CACHING). 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	1	0	1	2	3	4	5	6	7	8	9	2	0	1	2	3	4	5	6	7	8	9	3	0	1
Timeout																																		
Flags																																		
Reserved																																		
...																																		
CreateGuid																																		
...																																		
...																																		
...																																		

Timeout (4 bytes): The time, in milliseconds, for which the server reserves the handle after a failover, waiting for the client to reconnect. To let the server use a default timeout value, the client MUST set this field to 0.

Flags (4 bytes): This field MUST be constructed by using zero or more of the following values:

Value	Meaning
SMB2_DHANDLE_FLAG_PERSISTENT 0x00000002	A persistent handle is requested.

Reserved (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.

CreateGuid (16 bytes): A **GUID** that identifies the create request.

2.2.13.2.12 SMB2_CREATE_DURABLE_HANDLE_RECONNECT_V2

The SMB2_CREATE_DURABLE_HANDLE_RECONNECT_V2 context is specified when the client is attempting to reestablish a durable open as specified in section [3.2.4.4](#). The SMB2_CREATE_DURABLE_HANDLE_RECONNECT_V2 context is only valid for the SMB 3.0 dialect.

0	1	2	3	4	5	6	7	8	9	1	0	1	2	3	4	5	6	7	8	9	2	0	1	2	3	4	5	6	7	8	9	3	0	1
FileId																																		

...
...
...
CreateGuid
...
...
...
Flags

FileId (16 bytes): An [SMB2_FILEID](#) structure, as specified in section [2.2.14.1](#), for the open that is being reestablished.

CreateGuid (16 bytes): A unique ID that identifies the create request.

Flags (4 bytes): This field MUST be constructed using zero or more of the following values:

Value	Meaning
SMB2_DHANDLE_FLAG_PERSISTENT 0x00000002	A persistent handle is requested.

2.2.13.2.13 SMB2_CREATE_APP_INSTANCE_ID

The SMB2_CREATE_APP_INSTANCE_ID context is specified on an SMB2 CREATE Request when the client is supplying an identifier provided by an application. The SMB2_CREATE_APP_INSTANCE_ID context is only valid for the SMB 3.0 dialect. The client SHOULD also request a durable handle by using an [SMB2_CREATE_DURABLE_HANDLE_REQUEST_V2](#) or [SMB2_CREATE_DURABLE_HANDLE_RECONNECT_V2](#) create context.

0	1	2	3	4	5	6	7	8	9	1	0	1	2	3	4	5	6	7	8	9	2	0	1	2	3	4	5	6	7	8	9	3	0	1
StructureSize																Reserved																		
AppInstanceId																																		
...																																		
...																																		

StructureSize (2 bytes): The client MUST set this field to 20, indicating the size of this structure.

Reserved (2 bytes): This field MUST NOT be used and MUST be reserved. This field MUST be set to zero.

AppInstanceId (16 bytes): A unique ID that identifies an application instance.

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	1	0	1	2	3	4	5	6	7	8	9	2	0	1	2	3	4	5	6	7	8	9	3	0	1												
StructureSize								OplockLevel								Flags																														
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.[<45>](#)

Value	Meaning
SMB2_OPLOCK_LEVEL_NONE 0x00	No oplock was granted.
SMB2_OPLOCK_LEVEL_II 0x01	A level II oplock was granted.
SMB2_OPLOCK_LEVEL_EXCLUSIVE 0x08	An exclusive oplock was granted.
SMB2_OPLOCK_LEVEL_BATCH 0x09	A batch oplock was granted.
OPLOCK_LEVELLEASE 0xFF	A lease is requested. If set, the response packet MUST contain an SMB2_CREATE_RESPONSE_LEASE create context.

Flags (1 byte): If the server implements the SMB 3.0 dialect, this field MUST be constructed using the following value. Otherwise, this field MUST NOT be used and MUST be reserved.

Value	Meaning
SMB2_CREATE_FLAG_REPARSEPOINT 0x01	When set, indicates the last portion of the file path is a reparse point.

CreateAction (4 bytes): The action taken in establishing the open. This field MUST contain one of the following values.[<46>](#)

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.[<47>](#)

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	1	0	1	2	3	4	5	6	7	8	9	2	0	1	2	3	4	5	6	7	8	9	3	0	1
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 can 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	1	0	1	2	3	4	5	6	7	8	9	2	0	1	2	3	4	5	6	7	8	9	3	0	1
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

The server responds to an [SMB2_CREATE_DURABLE_HANDLE_RECONNECT](#) request as specified in section [3.3.5.9.7](#).

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	1	0	1	2	3	4	5	6	7	8	9	2	0	1	2	3	4	5	6	7	8	9	3	0	1
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_APP_INSTANCE_ID

The SMB2_CREATE_APP_INSTANCE_ID request has no associated [SMB2_CREATE_CONTEXT Response](#).

2.2.14.2.7 SMB2_CREATE_ALLOCATION_SIZE

The SMB2_CREATE_ALLOCATION_SIZE request does not generate an [SMB2_CREATE_CONTEXT Response](#).

2.2.14.2.8 SMB2_CREATE_TIMEWARP_TOKEN

The SMB2_CREATE_TIMEWARP_TOKEN request does not generate an [SMB2_CREATE_CONTEXT Response](#).

2.2.14.2.9 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	1	0	1	2	3	4	5	6	7	8	9	2	0	1	2	3	4	5	6	7	8	9	3	0	1
DiskIDBuffer																																		
...																																		
...																																		
...																																		
...																																		
...																																		
...																																		

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

2.2.14.2.10 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	1	0	1	2	3	4	5	6	7	8	9	2	0	1	2	3	4	5	6	7	8	9	3	0	1
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 the SMB 2.1 or SMB 3.0 dialect, this field MUST be set to zero or more of the following values. 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.

2.2.14.2.11 SMB2_CREATE_RESPONSELEASE_V2

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. The SMB2_CREATE_RESPONSELEASE_V2 context is only valid for the SMB 3.0 dialect.

0	1	2	3	4	5	6	7	8	9	1	0	1	2	3	4	5	6	7	8	9	2	0	1	2	3	4	5	6	7	8	9	3	0	1
LeaseKey																																		
...																																		
...																																		
...																																		
LeaseState																																		

	Flags
	LeaseDuration
	...
	ParentLeaseKey
	...
	...
	...
Epoch	Reserved

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 by using the following values.

Value	Meaning
SMB2_LEASE_NONE 0x00000000	No lease is granted.
SMB2_LEASE_READ_CACHING 0x00000001	A read caching lease is granted.
SMB2_LEASE_HANDLE_CACHING 0x00000002	A handle caching lease is granted.
SMB2_LEASE_WRITE_CACHING 0x00000004	A write caching lease is granted.

Flags (4 bytes): This field MUST be set to zero or the following value.

Value	Meaning
SMB2_LEASE_FLAG_BREAK_IN_PROGRESS 0x00000002	A break for the lease identified by the lease key is in progress.
SMB2_LEASE_FLAG_PARENT_LEASE_KEY_SET 0x00000004	When set, indicates that the ParentLeaseKey is set.

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

ParentLeaseKey (16 bytes): A unique key that identifies the owner of the lease for the parent directory.

Epoch (2 bytes): A 16-bit unsigned integer incremented by the server on a lease state change.

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.

2.2.14.2.12 SMB2_CREATE_DURABLE_HANDLE_RESPONSE_V2

If the server succeeds in opening a durable handle to a file as requested by the client via the SMB2_CREATE_DURABLE_HANDLE_REQUEST_V2 (section 2.2.13.2.11), it MUST send an SMB2_CREATE_DURABLE_HANDLE_RESPONSE_V2 back to the client to inform the client that the handle is durable. The SMB2_CREATE_DURABLE_HANDLE_RESPONSE_V2 context is only valid for the SMB 3.0 dialect.

0	1	2	3	4	5	6	7	8	9	1	0	1	2	3	4	5	6	7	8	9	2	0	1	2	3	4	5	6	7	8	9	3	0	1
Timeout																																		
Flags																																		

Timeout (4 bytes): The server MUST set this field to the time, in milliseconds, it waits for the client to reconnect after a failover.

Flags (4 bytes): This field MUST be constructed using zero or more of the following values:

Value	Meaning
SMB2_DHANDLE_FLAG_PERSISTENT 0x00000002	A persistent handle is granted.

2.2.14.2.13 SMB2_CREATE_DURABLE_HANDLE_RECONNECT_V2

If the server succeeds in reconnecting a durable handle to a file as requested by the client via the SMB2_CREATE_DURABLE_HANDLE_RECONNECT_V2 (section 2.2.13.2.12), it MUST send an SMB2_CREATE_RESPONSELEASE or SMB2_CREATE_RESPONSELEASE_V2 back to the client to inform the client that the durable handle is reconnected. The SMB2_CREATE_DURABLE_HANDLE_RECONNECT_V2 context is only valid for the SMB 3.0 dialect.

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	1	0	1	2	3	4	5	6	7	8	9	2	0	1	2	3	4	5	6	7	8	9	3	0	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	1	0	1	2	3	4	5	6	7	8	9	2	0	1	2	3	4	5	6	7	8	9	3	0	1
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 either zero or the following value:

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	1	0	1	2	3	4	5	6	7	8	9	2	0	1	2	3	4	5	6	7	8	9	3	0	1
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	1	0	1	2	3	4	5	6	7	8	9	2	0	1	2	3	4	5	6	7	8	9	3	0	1
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	1	0	1	2	3	4	5	6	7	8	9	2	0	1	2	3	4	5	6	7	8	9	3	0	1
StructureSize																Padding																		
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): For SMB 2.002 and 2.1 dialects, 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. For the SMB 3.0 dialect, this field MUST contain exactly one of the following values:

Value	Meaning
SMB2_CHANNEL_NONE 0x00000000	No channel information is present in the request. The ReadChannelInfoOffset and ReadChannelInfoLength fields MUST be set to 0 by the client and MUST be ignored by the server.
SMB2_CHANNEL_RDMA_V1 0x00000001	One or more SMB_DIRECT_BUFFER_DESCRIPTOR_V1 structures as specified in [MS-SMBD] section 2.2.3.1 are present in the channel information specified by ReadChannelInfoOffset and ReadChannelInfoLength fields.

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): For the SMB 2.002 and 2.1 dialects, 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. For the SMB 3.0 dialect, it contains the offset, in bytes, from the beginning of the SMB2 header to the channel data as specified by the **Channel** field of the request.

ReadChannelInfoLength (2 bytes): For the SMB 2.002 and 2.1 dialects, 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. For the SMB 3.0 dialect, it contains the length, in bytes, of the channel data as specified by the **Channel** field of the request.

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	1	0	1	2	3	4	5	6	7	8	9	2	0	1	2	3	4	5	6	7	8	9	3	0	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.

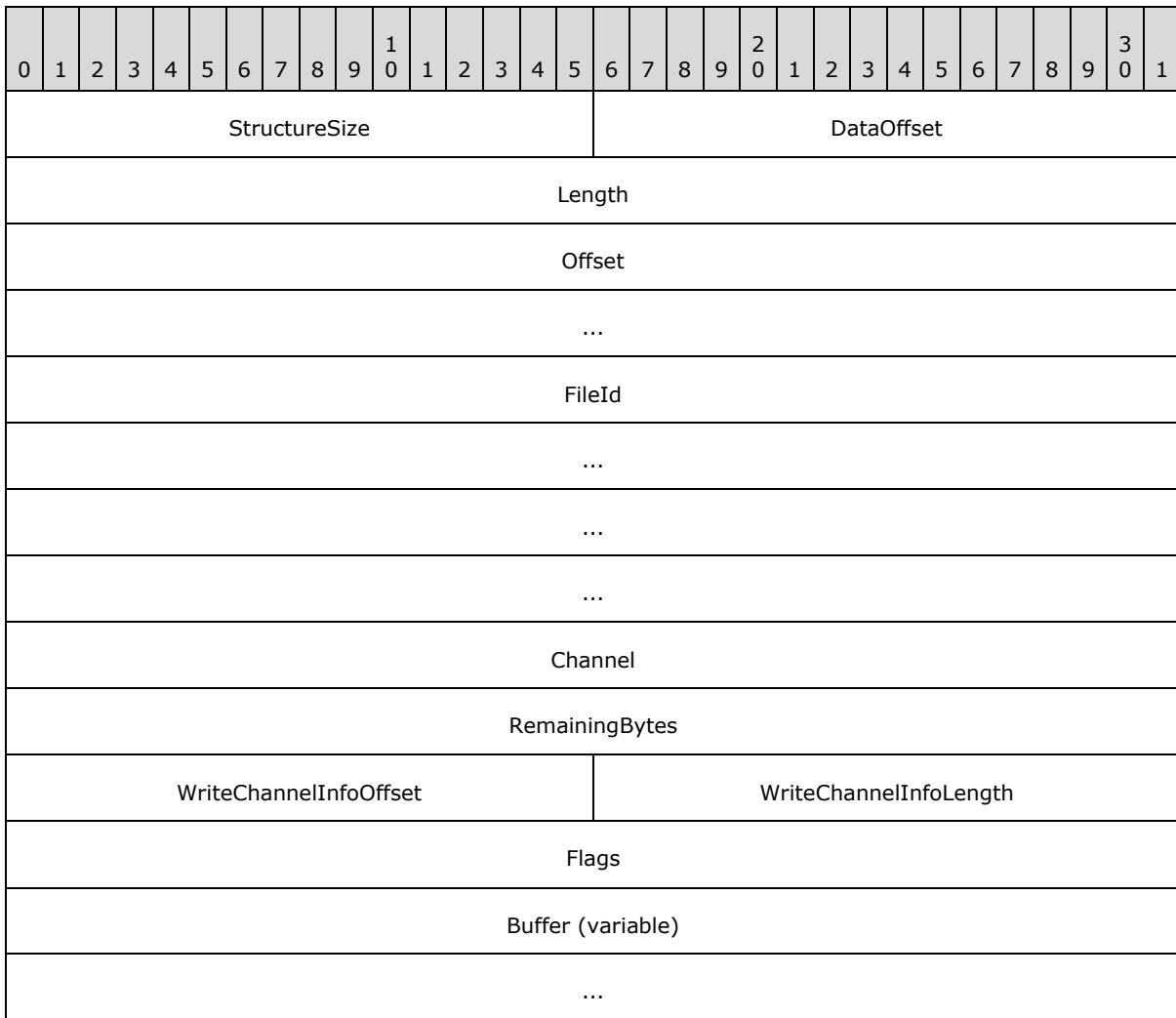
DataRemaining (4 bytes): This field MUST NOT be used and MUST be reserved. The server MUST set this field to 0, and the client MUST ignore it on receipt.

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:



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): For the SMB 2.002 and 2.1 dialects, 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. For the SMB 3.0 dialect, this field MUST contain exactly one of the following values:

Value	Meaning
SMB2_CHANNEL_NONE 0x00000000	No channel information is present in the request. The WriteChannelInfoOffset and WriteChannelInfoLength fields MUST be set to zero by the client and MUST be ignored by the server.
SMB2_CHANNEL_RDMA_V1 0x00000001	One or more SMB_DIRECT_BUFFER_DESCRIPTOR_V1 structures as specified in [MS-SMBD] section 2.2.3.1 are present in the channel information specified by WriteChannelInfoOffset and WriteChannelInfoLength fields.

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): For the SMB 2.002 and 2.1 dialects, 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. For the SMB 3.0 dialect, it contains the offset, in bytes, from the beginning of the SMB2 header to the channel data as described by the **Channel** field of the request.

WriteChannelInfoLength (2 bytes): For the SMB 2.002 and SMB 2.1 dialects, 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. For the SMB 3.0 dialect, it contains the offset, in bytes, from the beginning of the SMB2 header to the channel data as described by the **Channel** field of the request.

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 not valid for the SMB 2.002 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	1	0	1	2	3	4	5	6	7	8	9	2	0	1	2	3	4	5	6	7	8	9	3	0	1
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 message 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	1	0	1	2	3	4	5	6	7	8	9	2	0	1	2	3	4	5	6	7	8	9	3	0	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.

OlockLevel (1 byte): The server MUST set this to the maximum value of the **OlockLevel** 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. [<48>](#)

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 notification is not valid for the SMB 2.002 dialect. This message 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	1	0	1	2	3	4	5	6	7	8	9	2	0	1	2	3	4	5	6	7	8	9	3	0	1
StructureSize																NewEpoch																		
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.

NewEpoch (2 bytes): A 16-bit unsigned integer indicating a lease state change by the server. This field is only valid for a server implementing the SMB 3.0 dialect.

For the SMB 2.002 and SMB 2.1 dialects, 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 by using zero or more of the following values.

Value	Meaning
SMB2_NOTIFY_BREAK_LEASE_FLAG_ACK_REQUIRED 0x01	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.[<49>](#)

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.

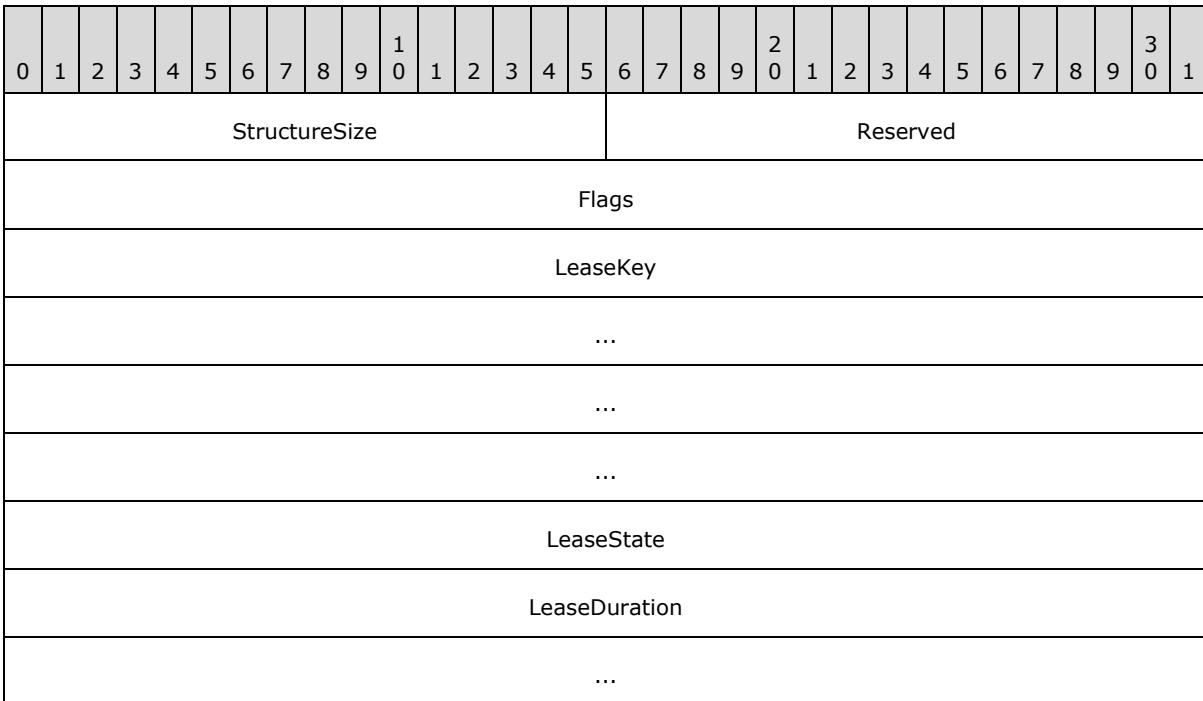
Reserved2 (4 bytes): This field MUST NOT be used and MUST be reserved. The client MUST set this to 0, and the server MUST ignore it on receipt.

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 acknowledgment is not valid for the SMB 2.002 dialect. The server responds to a **lease break** acknowledgment with an SMB2 [Lease Break Response](#).



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. [<50>](#) This field MUST be constructed using the following values:

Value	Meaning
SMB2_LEASE_NONE 0x00	No lease is granted.
SMB2_LEASE_READ_CACHING	A read caching lease is accepted.

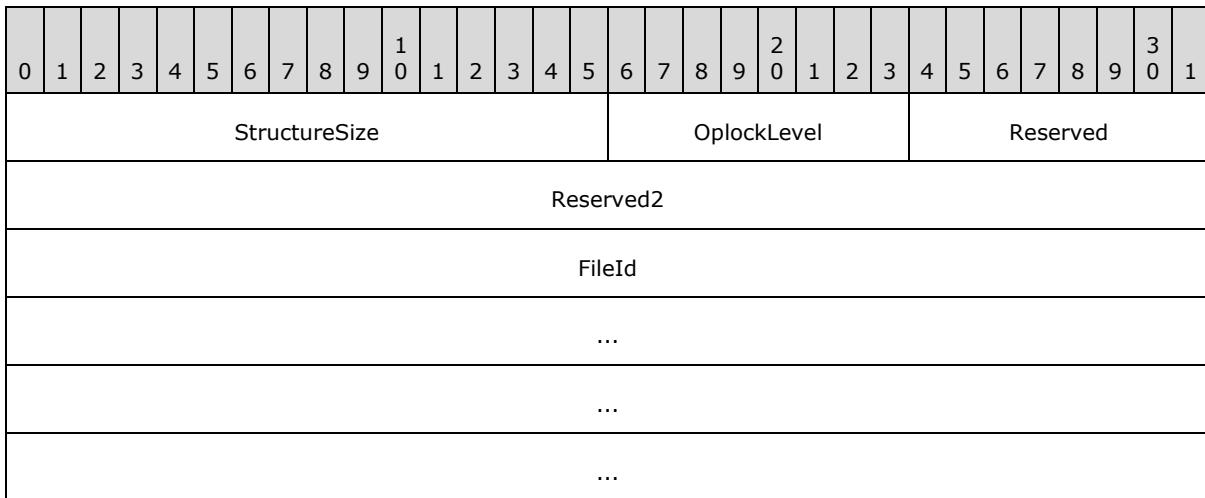
Value	Meaning
0x01	
SMB2_LEASE_HANDLE_CACHING 0x02	A handle caching lease is accepted.
SMB2_LEASE_WRITE_CACHING 0x04	A write caching lease is accepted.

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:



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 response is not valid for the SMB 2.002 dialect.

0	1	2	3	4	5	6	7	8	9	1	0	1	2	3	4	5	6	7	8	9	2	0	1	2	3	4	5	6	7	8	9	3	0	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:

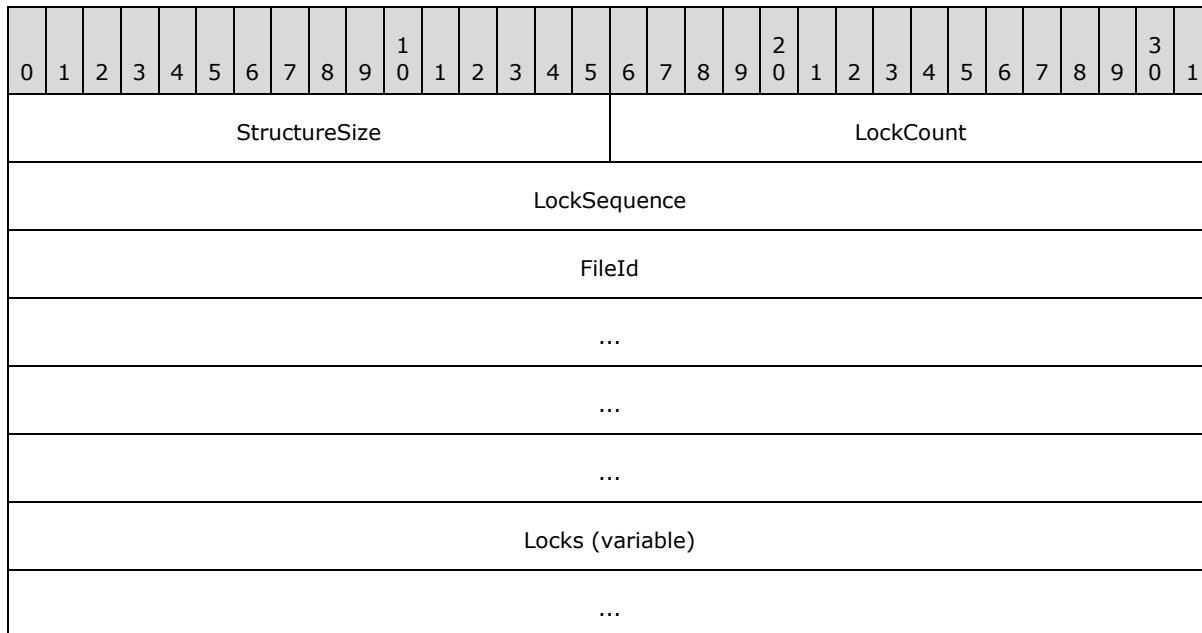
Value	Meaning
SMB2_LEASE_NONE 0x00	No lease is granted.
SMB2_LEASE_READ_CACHING 0x01	A read caching lease is granted.
SMB2_LEASE_HANDLE_CACHING 0x02	A handle caching lease is granted.
SMB2_LEASE_WRITE_CACHING 0x04	A write caching lease is granted.

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. It is the client's responsibility to locally resolve lock conflicts across multiple processes on the same client, if any such conflicts exist.



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.

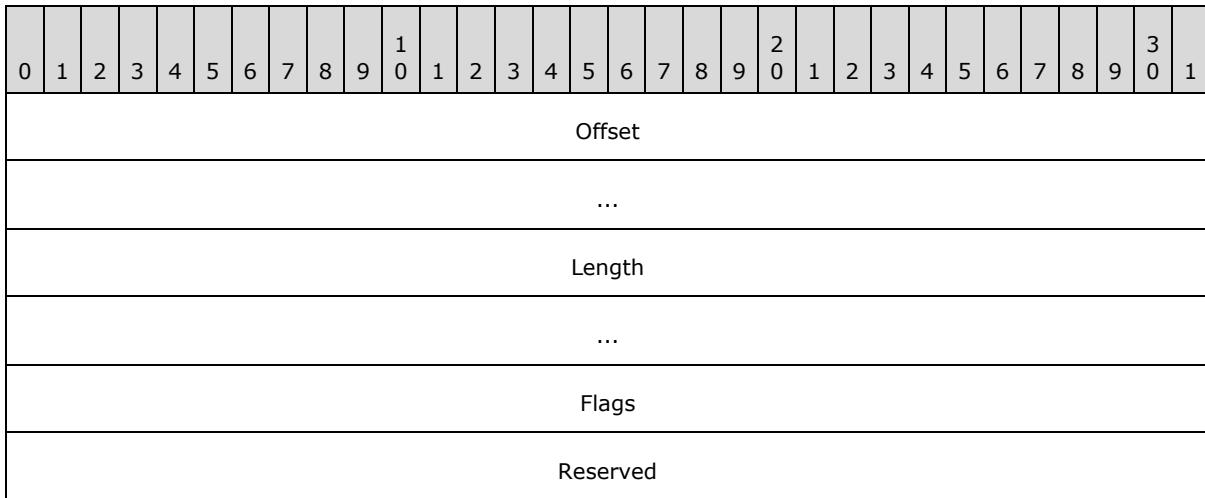
LockSequence (4 bytes): In the SMB 2.002 dialect, this field is unused and MUST be reserved. The client MUST set this to 0, and the server MUST ignore it on receipt. In all other dialects, this field indicates a value that identifies a lock or unlock request uniquely across all lock or unlock requests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



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.

Value	Meaning
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	1	0	1	2	3	4	5	6	7	8	9	2	0	1	2	3	4	5	6	7	8	9	3	0	1
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	1	0	1	2	3	4	5	6	7	8	9	2	0	1	2	3	4	5	6	7	8	9	3	0	1
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	1	0	1	2	3	4	5	6	7	8	9	2	0	1	2	3	4	5	6	7	8	9	3	0	1
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	1	0	1	2	3	4	5	6	7	8	9	2	0	1	2	3	4	5	6	7	8	9	3	0	1
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	1	0	1	2	3	4	5	6	7	8	9	2	0	1	2	3	4	5	6	7	8	9	3	0	1
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

Name	Value
FSCTL_SRV_REQUEST_RESUME_KEY	0x00140078
FSCTL_SRV_READ_HASH	0x001441bb
FSCTL_SRV_COPYCHUNK_WRITE	0x001480F2
FSCTL_LMR_REQUEST_RESILIENCY	0x001401D4
FSCTL_QUERY_NETWORK_INTERFACE_INFO	0x001401FC
FSCTL_SET_REPARSE_POINT	0x000900A4
FSCTL_DFS_GET_REFERRALS_EX	0x000601B0
FSCTL_FILE_LEVEL_TRIM	0x00098208
FSCTL_VALIDATE_NEGOTIATE_INFO	0x00140204

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.

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.[<51>](#)

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.[<52>](#)

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	1	0	1	2	3	4	5	6	7	8	9	2	0	1	2	3	4	5	6	7	8	9	3	0	1
SourceKey																																		
...																																		
...																																		
...																																		
...																																		
...																																		
...																																		
ChunkCount																																		
Reserved																																		
Chunks (variable)																																		
...																																		

SourceKey (24 bytes): A key, obtained from the server in a [SRV REQUEST RESUME KEY Response \(section 2.2.32.3\)](#), that represents the source file for the copy.

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

Reserved (4 bytes): This field MUST 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	1	0	1	2	3	4	5	6	7	8	9	2	0	1	2	3	4	5	6	7	8	9	3	0	1
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.[<53>](#)

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 and 3.0 dialects. 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	1	0	1	2	3	4	5	6	7	8	9	2	0	1	2	3	4	5	6	7	8	9	3	0	1
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 one of the following values:

Value	Meaning
SRV_HASH_VER_1 0x00000001	Branch cache version 1.
SRV_HASH_VER_2 0x00000002	Branch cache version 2. This value is only applicable for SMB dialect 3.0.

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

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.
SRV_HASH_RETRIEVE_FILE_BASED 0x00000002	The Offset field in the SRV_READ_HASH request is relative to the beginning of the file indicated by the FileId field in the IOCTL request. This value is only applicable for SMB dialect 3.0.

Length (4 bytes): If **HashRetrievalType** is SRV_HASH_RETRIEVE_HASH_BASED, this value is the maximum length, in bytes, of the hash data to be returned in the SRV_READ_HASH response to the client. If **HashRetrievalType** is SRV_HASH_RETRIEVE_FILE_BASED, this value is the maximum length, in bytes, of the file data for which the hash information is to be retrieved and returned in the SRV_READ_HASH response to the client.

Offset (8 bytes): If **HashRetrievalType** is SRV_HASH_RETRIEVE_HASH_BASED, this value is the offset of the data to be retrieved, in bytes, from the beginning of the Content Information File. If **HashRetrievalType** is SRV_HASH_RETRIEVE_FILE_BASED, this value is the offset in the file for which the hash information is to be retrieved.

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. This request is not valid for the SMB 2.002 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	1	0	1	2	3	4	5	6	7	8	9	2	0	1	2	3	4	5	6	7	8	9	3	0	1
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.

2.2.31.4 VALIDATE_NEGOTIATE_INFO Request

The VALIDATE_NEGOTIATE_INFO request packet is sent to the server by the client in an SMB2 IOCTL Request FSCTL_VALIDATE_NEGOTIATE_INFO to request validation of a previous SMB 2 NEGOTIATE. The request is valid for clients and servers which implement the SMB 3.0 dialect.

0	1	2	3	4	5	6	7	8	9	1	0	1	2	3	4	5	6	7	8	9	2	0	1	2	3	4	5	6	7	8	9	3	0	1
Capabilities																																		
Guid																																		
...																																		
...																																		
...																																		
SecurityMode																DialectCount																		
Dialects (variable)																																		
...																																		

Capabilities (4 bytes): The **Capabilities** of the client.

Guid (16 bytes): The **ClientGuid** of the client.

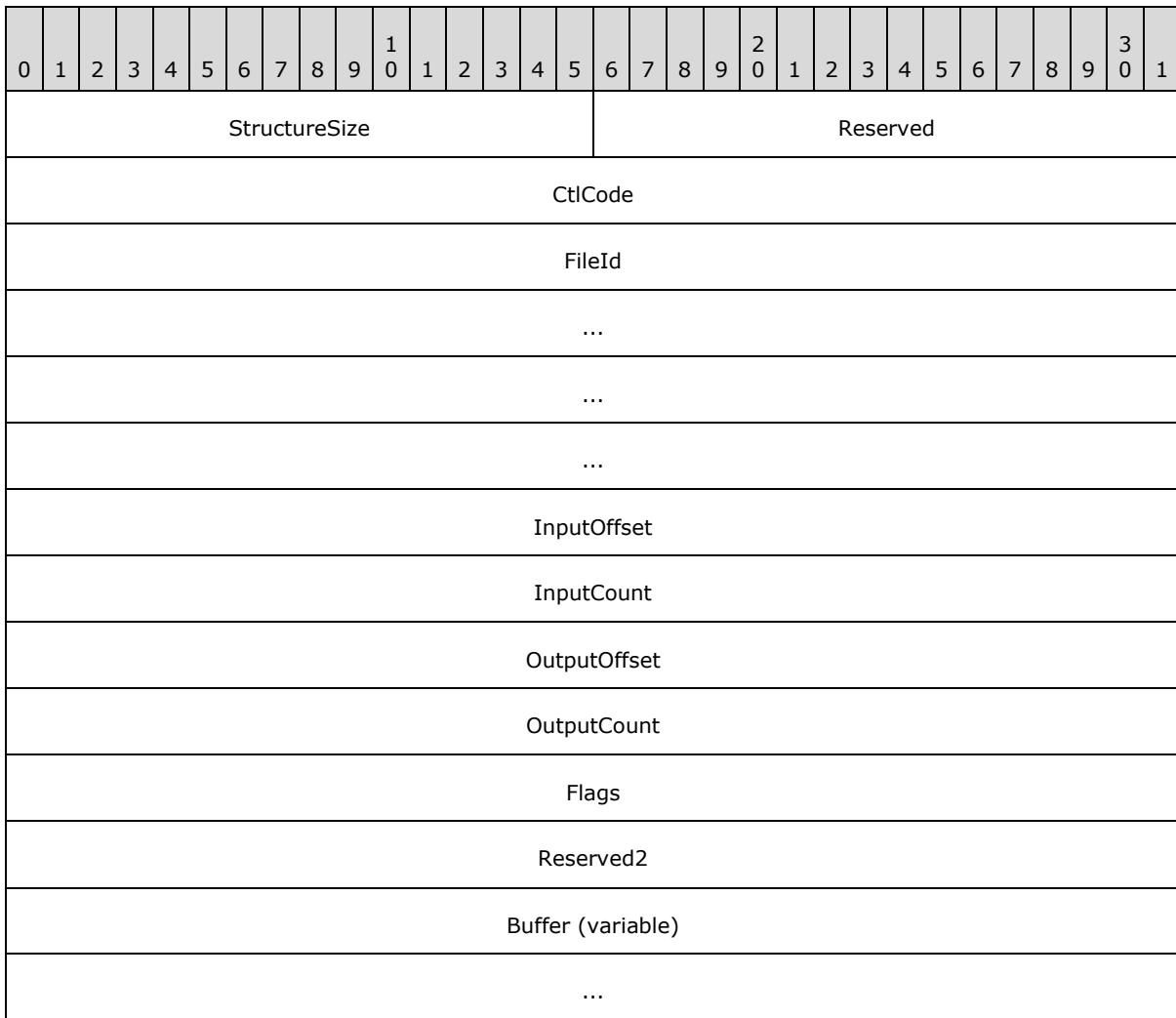
SecurityMode (2 bytes): The **SecurityMode** of the client.

DialectCount (2 bytes): The number of entries in the **Dialects** field.

Dialects (variable): The list of SMB2 dialects supported by the client. These entries SHOULD contain only the 2-byte **Dialects** values defined in section [2.2.3](#).

2.2.32 SMB2 IOCTL Response

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



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

Reserved (2 bytes): 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 0xFFFFFFFFFFFFFFF 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 [<54>](#) 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 [<55>](#) 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	1	0	1	2	3	4	5	6	7	8	9	2	0	1	2	3	4	5	6	7	8	9	3	0	1
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, [<56>](#) 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	1	0	1	2	3	4	5	6	7	8	9	2	0	1	2	3	4	5	6	7	8	9	3	0	1
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	1	0	1	2	3	4	5	6	7	8	9	2	0	1	2	3	4	5	6	7	8	9	3	0	1
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.[<57>](#)

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](#).

2.2.32.4.1 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 and 3.0 dialects.

Content information follows this header at an offset indicated by the **HashBlobOffset** field, if **HashVersion** is set to SRV_HASH_VER_1, the Content Information data structure is as specified in [\[MS-PCCRC\]](#) section 2.3; if **HashVersion** is set to SRV_HASH_VER_2, the Content Information data structure is as specified in [\[MS-PCCRC\]](#) section 2.4.

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

HashType (4 bytes): The hash type 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 used 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 constructed using one of the following values.

Value	Meaning
SRV_HASH_VER_1 0x00000001	Branch cache version 1.
SRV_HASH_VER_2 0x00000002	Branch cache version 2. This value is only applicable for servers that implement the SMB 3.0 dialect.

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**.[<58>](#)

2.2.32.4.2 SRV_HASH_RETRIEVE_HASH_BASED

If the **HashRetrievalType** in the request is SRV_HASH_RETRIEVE_HASH_BASED the SRV_READ_HASH response MUST be formatted as follows:

0	1	2	3	4	5	6	7	8	9	1	0	1	2	3	4	5	6	7	8	9	2	0	1	2	3	4	5	6	7	8	9	3	0	1
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, as specified in [\[MS-PCCRC\]](#) section 2.3.

2.2.32.4.3 SRV_HASH_RETRIEVE_FILE_BASED

This response is valid for servers that implement the SMB 3.0 dialect. If the **HashRetrievalType** in the request is SRV_HASH_RETRIEVE_FILE_BASED, the SRV_READ_HASH response MUST be formatted as follows:

0	1	2	3	4	5	6	7	8	9	1	0	1	2	3	4	5	6	7	8	9	2	0	1	2	3	4	5	6	7	8	9	3	0	1
FileDataOffset																																		
...																																		
FileDataLength																																		
...																																		
BufferLength																																		
Reserved																																		
Buffer (variable)																																		
...																																		

FileDataOffset (8 bytes): File data offset corresponding to the start of the hash data returned.

FileDataLength (8 bytes): The length, in bytes, starting from the **FileDataOffset** that is covered by the hash data returned.

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 zero, and the client MUST ignore it on receipt.

Buffer (variable): A variable-length buffer that contains the retrieved portion of the Content Information File, as specified in [\[MS-PCCRC\]](#) section 2.4.

2.2.32.5 NETWORK_INTERFACE_INFO Response

The NETWORK_INTERFACE_INFO is returned to the client by the server in an SMB2 IOCTL response for FSCTL_QUERY_NETWORK_INTERFACE_INFO request. The interface structure is defined as following.

0	1	2	3	4	5	6	7	8	9	1	0	1	2	3	4	5	6	7	8	9	2	0	1	2	3	4	5	6	7	8	9	3	0	1
Next																																		
IfIndex																																		
Capability																																		
Reserved																																		

LinkSpeed
...
SockAddr_Storage
...
...
...
...
...
...
...
...
...
...
...
...
...
(SockAddr_Storage cont'd for 24 rows)

Next (4 bytes): The offset, in bytes, from the beginning of this structure to the beginning of a subsequent 8-byte aligned network interface. This field MUST be set to zero if there are no subsequent network interfaces.

IfIndex (4 bytes): This field specifies the network interface index.

Capability (4 bytes): This field specifies the capabilities of the network interface. This field MUST be constructed using zero or more of the following values:

Value	Meaning
RSS_CAPABLE 0x00000001	When set, specifies that the interface is RSS-capable.
RDMA_CAPABLE 0x00000002	When set, specifies that the interface is RDMA-capable.

Reserved (4 bytes): This field MUST be set to zero and the client MUST ignore it on receipt.

LinkSpeed (8 bytes): The field specifies the speed of the network interface in bits per second.

SockAddr_Storage (128 bytes): The field describes socket address information as specified in section [2.2.32.5.1](#).

2.2.32.5.1 SOCKADDR_STORAGE

Socket Address Information is a 128-byte structure formatted as follows:

0	1	2	3	4	5	6	7	8	9	1	0	1	2	3	4	5	6	7	8	9	2	0	1	2	3	4	5	6	7	8	9	3	0	1
Family																Buffer (variable)																		
...																																		
Reserved (variable)																...																		
...																...																		

Family (2 bytes): Address family of the socket. This field MUST contain one of the following values:

Value	Meaning
InterNetwork 0x0002	When set, indicates an IPv4 address in the socket.
InterNetworkV6 0x0017	When set, indicates an IPv6 address in the socket.

Buffer (variable): A variable-length buffer that contains the socket address information. If the value of the field **Family** is 0x0002, this field MUST be interpreted as **SOCKADDR_IN**, specified in [2.2.32.5.1.1](#). Otherwise, if the value of the field **Family** is 0x0017, this field MUST be interpreted as **SOCKADDR_IN6**, specified in [2.2.32.5.1.2](#).

Reserved (variable): The remaining bytes within the size of **SOCKADDR_STORAGE** structure (128 bytes) MUST NOT be used and MUST be reserved. The server SHOULD set this to zero, and the client MUST ignore it on receipt.

2.2.32.5.1.1 SOCKADDR_IN

This socket address information is a 14-byte structure formatted as follows. All fields in this structure are in network byte order.

0	1	2	3	4	5	6	7	8	9	1	0	1	2	3	4	5	6	7	8	9	2	0	1	2	3	4	5	6	7	8	9	3	0	1
Port																IPv4Address																		
...																Reserved																		
...																...																		
...																...																		

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

IPv4Address (4 bytes): IPv4 address.

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

2.2.32.5.1.2 SOCKADDR_IN6

This socket address information is a 26-byte structure formatted as follows. All fields in this structure are in network byte order.

0	1	2	3	4	5	6	7	8	9	1	0	1	2	3	4	5	6	7	8	9	2	0	1	2	3	4	5	6	7	8	9	3	0	1
Port																FlowInfo																		
...																IPv6Address																		
...																...																		
...																ScopeId																		
...																...																		

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

FlowInfo (4 bytes): The server SHOULD set this field to zero, and the client MUST ignore it on receipt.

IPv6Address (16 bytes): IPv6 address.

ScopeId (4 bytes): The server SHOULD set this field to zero, and the client MUST ignore it on receipt.

2.2.32.6 VALIDATE_NEGOTIATE_INFO Response

The VALIDATE_NEGOTIATE_INFO response is returned to the client by the server in an SMB2 IOCTL response for FSCTL_VALIDATE_NEGOTIATE_INFO request. The response is valid for servers which implement the SMB 3.0 dialect, and optional for others.

0	1	2	3	4	5	6	7	8	9	1	0	1	2	3	4	5	6	7	8	9	2	0	1	2	3	4	5	6	7	8	9	3	0	1
Capabilities																																		
Guid																																		
...																																		

	...
	...
SecurityMode	Dialect

Capabilities (4 bytes): The **Capabilities** of the server.

Guid (16 bytes): The **ServerGuid** of the server.

SecurityMode (2 bytes): The **SecurityMode** of the server.

Dialect (2 bytes): The SMB2 **dialect** in use by the server on the connection.

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	1	0	1	2	3	4	5	6	7	8	9	2	0	1	2	3	4	5	6	7	8	9	3	0	1															
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.

FileInfoClass (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 <59> 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.

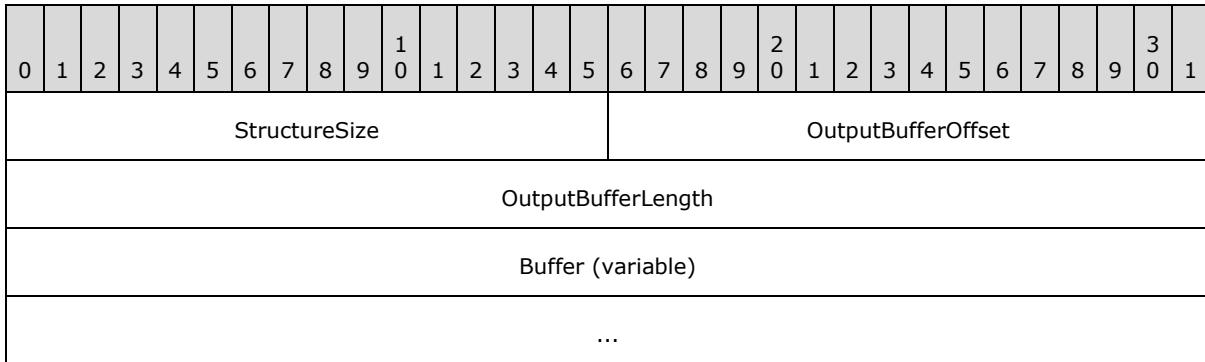
FileNameLength (2 bytes): The length, in bytes, of the search pattern. This field MUST be 0 if no search pattern is provided.

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.<60>](#)

2.2.34 SMB2 QUERY_DIRECTORY Response

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



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

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

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

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

2.2.35 SMB2 CHANGE_NOTIFY Request

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

0	1	2	3	4	5	6	7	8	9	1	0	1	2	3	4	5	6	7	8	9	2	0	1	2	3	4	5	6	7	8	9	3	0	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 either zero or the following value:

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] section 2.6 .

Value	Meaning
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 change. 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	1	0	1	2	3	4	5	6	7	8	9	2	0	1	2	3	4	5	6	7	8	9	3	0	1
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.

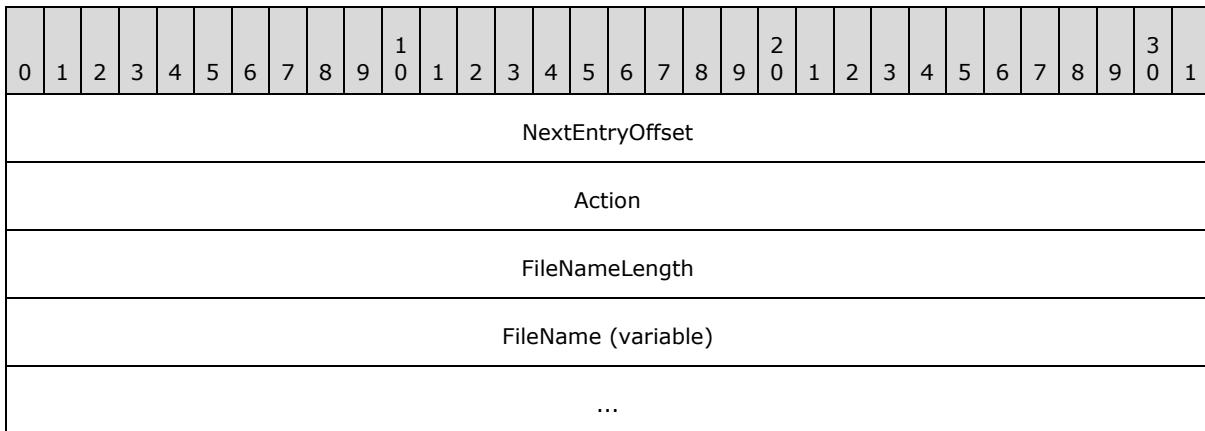
OutputBufferOffset (2 bytes): The offset, in bytes, from the beginning of the SMB2 header to the change information being returned.

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:



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.

Value	Meaning
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.

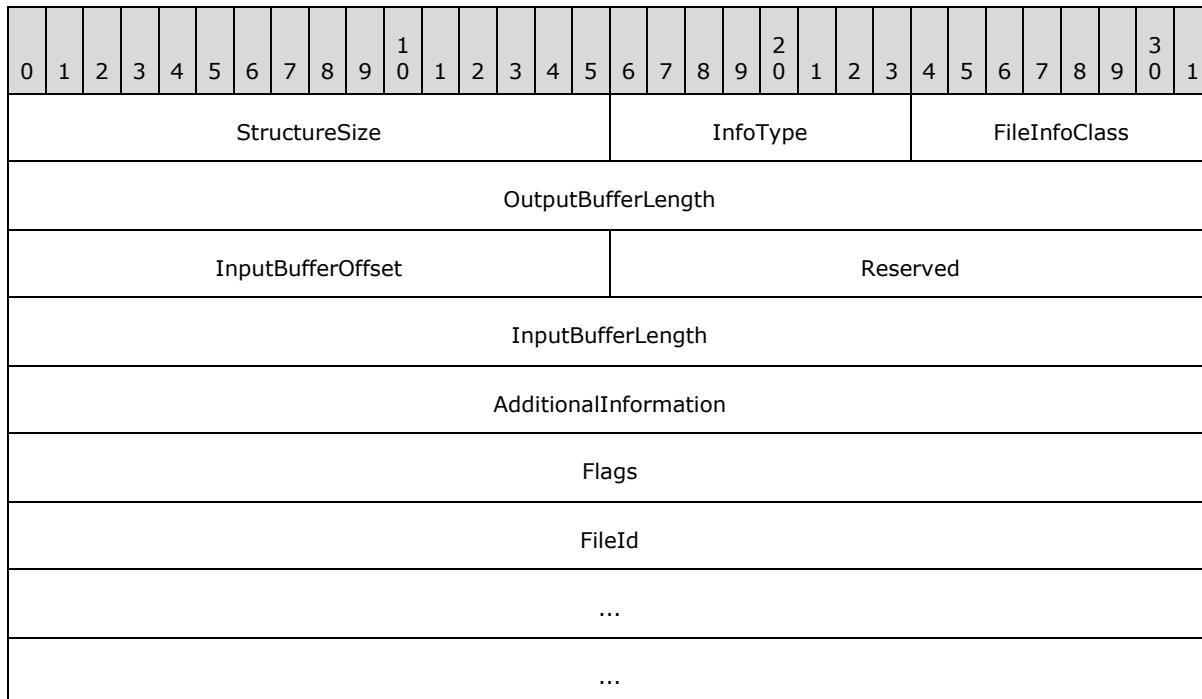
If two or more files have been renamed, then the corresponding FILE_NOTIFY_INFORMATION entries for each file rename MUST be consecutive in this response, in order for the client to make the correct correspondence between old and new names.

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:



...
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
- FileFsObjectInformation
- FileFsSectorSizeInformation
- FileFsSizeInformation
- FileFsVolumeInformation

For security queries, this field MUST be set to 0. For quota queries, this field SHOULD [`<61>`](#) 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 [`<62>`](#) 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 SHOULD be set to 0 and the server MUST ignore it on receipt.

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 0x00000001	Restart the scan for EAs from the beginning.
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.[<63>](#)

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	1	0	1	2	3	4	5	6	7	8	9	2	0	1	2	3	4	5	6	7	8	9	3	0	1		
ReturnSingle					RestartScan					Reserved																										
SidListLength																																				
StartSidLength																																				
StartSidOffset																																				
SidBuffer (variable)																																				
...																																				

ReturnSingle (1 byte): A Boolean value, where zero represents FALSE and nonzero represents TRUE.[<64>](#) 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.[<65>](#) 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.

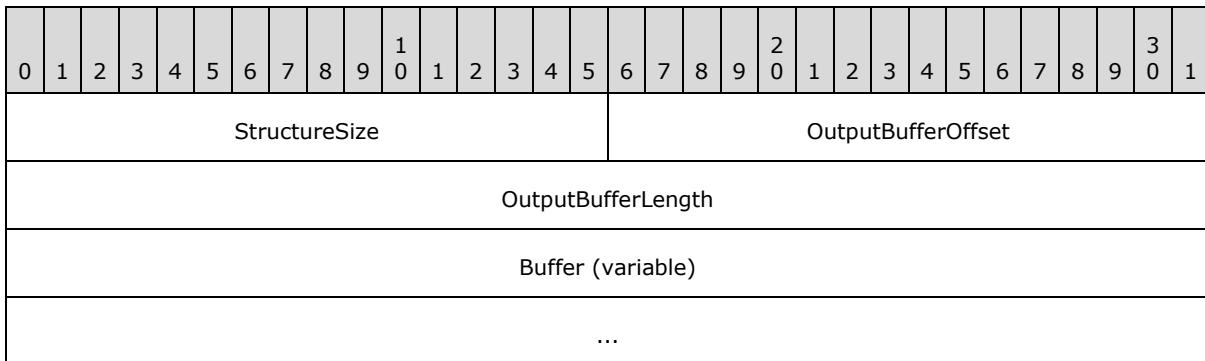
StartSidOffset (4 bytes): The offset, in bytes, from the start of the **SidBuffer** field to the SID when sent in format 2 as defined in the **SidBuffer** field, or zero in all other cases.

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.[<66>](#)

2.2.38 SMB2 QUERY_INFO Response

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



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

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

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

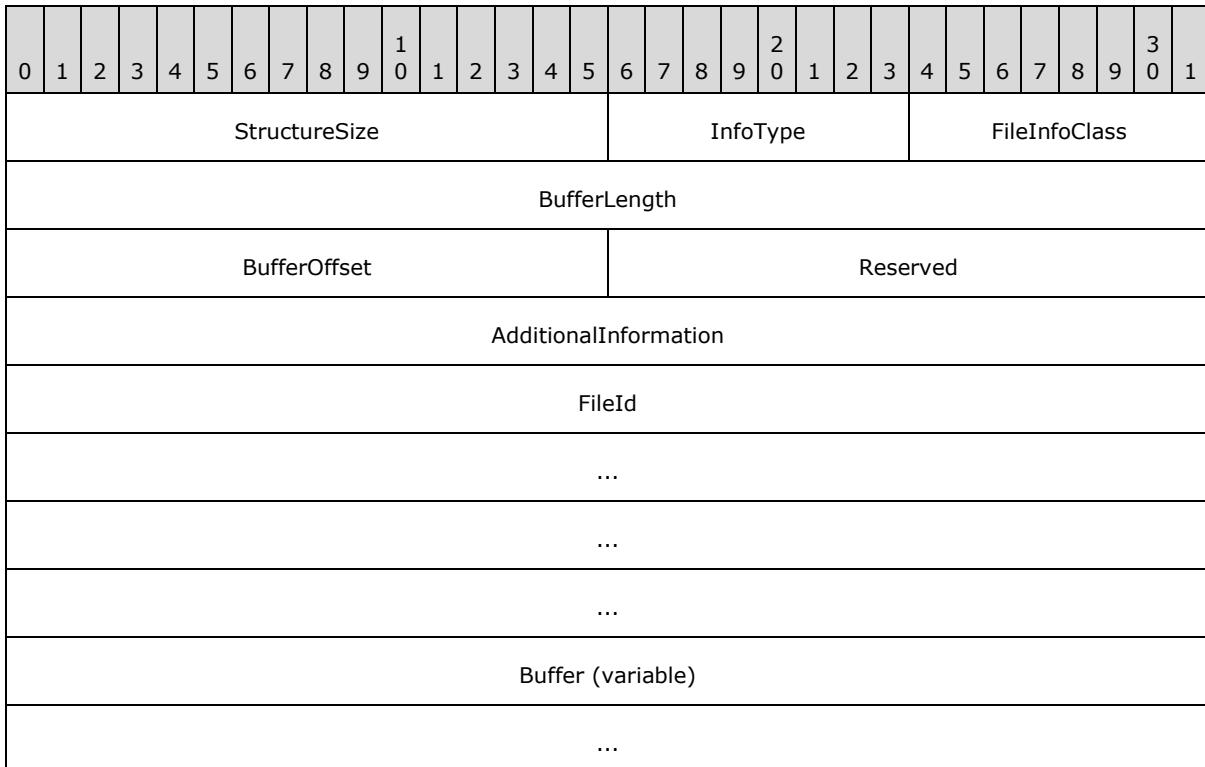
Buffer (variable): A variable-length buffer that contains the information that is returned in the response, as described by the **OutputBufferOffset** and **OutputBufferLength** fields. Buffer format depends on **InfoType** and **AdditionalInformation**, as follows.

InfoType	AdditionalInformation	Buffer format specification
SMB2_0_INFO_FILE	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.

InfoType	AdditionalInformation	Buffer format specification
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.



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.

Value	Meaning
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. [<67>](#)

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.

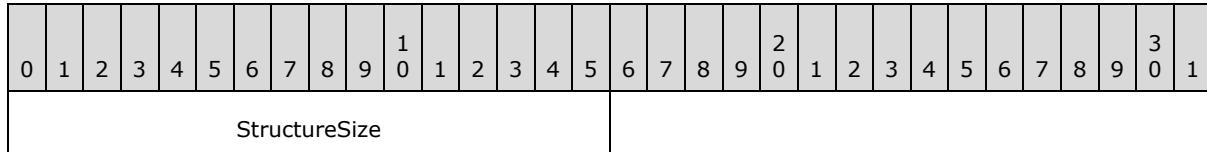
FileId (16 bytes): An [SMB2_FILEID](#) identifier of the file or named pipe on which to perform the set. Set operations for underlying object store and quota information are directed to the volume on which the file resides.

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: 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.

2.2.40 SMB2_SET_INFO Response

The SMB2 SET_INFO Response packet is sent by the server in response to an [SMB2_SET_INFO Request \(section 2.2.39\)](#) to notify the client that its request has been successfully processed. This response consists of an SMB2 header, as specified in section [2.2.1](#), followed by this response structure:



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

2.2.41 SMB2 TRANSFORM_HEADER

The SMB2 Transform Header is used by the client or server when sending encrypted messages. The SMB2 TRANSFORM_HEADER is only valid for the SMB 3.0 dialect.

0	1	2	3	4	5	6	7	8	9	1	0	1	2	3	4	5	6	7	8	9	2	0	1	2	3	4	5	6	7	8	9	3	0	1													
ProtocolId																																															
Signature																																															
...																																															
...																																															
...																																															
Nonce																																															
...																																															
...																																															
...																																															
OriginalMessageSize																																															
Reserved																EncryptionAlgorithm																															
SessionId																																															
...																																															

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

Signature (16 bytes): The 16-byte signature of the encrypted message generated by using **Session.EncryptionKey**.

Nonce (16 bytes): An implementation-specific value assigned for every encrypted message. This MUST NOT be reused for all encrypted messages within a session.

OriginalMessageSize (4 bytes): The size, in bytes, of the SMB2 message.

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

EncryptionAlgorithm (2 bytes): The algorithm used for encrypting the SMB2 message. This field MUST be set to one of the following values:

Value	Meaning
SMB2_ENCRYPTION_AES128_CCM 0x0001	The message is encrypted by using the AES128 algorithm.

SessionId (8 bytes): Uniquely identifies the established session for the command.

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 can 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.[<68>](#)

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 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. If **Connection.Dialect** is "3.000", the sender MUST compute a 16-byte hash using AES_CMAC-128 over the entire message, beginning with the SMB2 Header from step 1, and using the key provided. The AES_CMAC-128 is specified in [\[RFC4493\]](#). If the message is part of a compounded chain, any padding at the end of the message MUST be used in the hash computation. The sender MUST copy the 16-byte hash into the signature field of the SMB2 header.
3. If **Connection.Dialect** is "2.002" or "2.100", 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 key provided. 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. The first 16 bytes (the high-order portion) of the hash 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](#).

3.1.4.2 Generating Cryptographic Keys

This optional interface is applicable only for the SMB 3.0 dialect.

When cryptographic keys are to be generated by processing as specified in sections [3.2.5.3](#) and [3.3.5.5](#), the Key Derivation specification in [\[SP800-108\]](#) is used with the following inputs:

- The key to be used for key derivation.
- The string to be used as label.
- The length of the label string.
- The string to be used as the context.
- The length of the context string.

The cryptographic keys MUST be generated using the KDF algorithm in Counter Mode, as specified in [SP800-108] section 5.1, with 'r' value of 32 and 'L' value of 128 and by providing the inputs mentioned above. The PRF used in the key derivation MUST be HMAC-SHA256.

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 key to verify the signature. 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 5.
2. The receiver MUST zero out the 16-byte signature field in the SMB2 Header of the incoming message.
3. If **Connection.Dialect** is "3.000", the receiver MUST compute a 16-byte hash by using AES_CMAC-128 over the entire message, beginning with the SMB2 Header from step 2, and using the key provided. The AES_CMAC-128 is specified in [\[RFC4493\]](#). 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 **Connection.Dialect** is "2.002" or "2.100", the receiver MUST compute a 32-byte hash by using HMAC-SHA256 over the entire message, beginning with the SMB2 Header from step 2, and using the key provided. 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.
5. If the first 16 bytes (the high-order portion) of the computed signature from step 3 or step 4 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 [3.2.5.1.3](#). Determining when a server will verify a signature and the action taken on the result of verification is specified in section [3.3.5.2.4](#).

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 2.1 or SMB 3.0 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.

If the client implements the SMB 3.0 dialect, it MUST also implement the following:

EncryptionAlgorithmList: A list of strings containing the names of encryption algorithms supported by the client.

MaxDialect: The highest SMB2 dialect that the client implements. It MUST take the format of dialect values as specified in section [2.2.3](#).

RequireSecureNegotiate: A Boolean that, if set, indicates that the client requires validation of an SMB2 NEGOTIATE request.

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.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.[<69>](#)

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 the client implements the SMB 2.1 or SMB 3.0 dialects, it MUST also implement the following:

- **Connection.Dialect:** The dialect of SMB2 negotiated with the server. This value MUST be "2.002", "2.100", "3.000", or "Unknown".
- **Connection.SupportsFileLeasing:** A Boolean indicating whether the server supports file leasing functionality.
- **Connection.SupportsMultiCredit:** A Boolean indicating whether the server supports multi-credit operations.

If the client implements the SMB 3.0 dialect, it MUST also implement the following:

- **Connection.SupportsDirectoryLeasing:** A Boolean indicating whether the server supports directory leasing.
- **Connection.SupportsMultiChannel:** A Boolean indicating whether the server supports establishing multiple channels for sessions.
- **Connection.SupportsPersistentHandles:** A Boolean indicating whether the server supports persistent handles.
- **Connection.SupportsEncryption:** A Boolean indicating whether the SMB2 server supports encryption.

- **Connection.ClientCapabilities**: The capabilities sent by the client in the SMB2 NEGOTIATE Request on this connection, in a form that MUST follow the syntax as specified in section [2.2.3](#).
- **Connection.ServerCapabilities**: The capabilities received from the server in the SMB2 NEGOTIATE Response on this connection, in a form that MUST follow the syntax as specified in section [2.2.4](#).
- **Connection.ClientSecurityMode**: The security mode sent by the client in the SMB2 NEGOTIATE request on this connection, in a form that MUST follow the syntax as specified in section [2.2.3](#).
- **Connection.ServerSecurityMode**: The security mode received from the server in the SMB2 NEGOTIATE response on this connection, in a form that MUST follow the syntax as specified in section [2.2.4](#).

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.SigningRequired: A Boolean that, if set, indicates that all of the messages for this session MUST be signed.

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.

Session.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**.

If the client implements the SMB 3.0 dialect, it MUST also implement the following:

Session.ChannelList: A list of channels, as specified in section [3.2.1.8](#).

Session.ChannelSequence: A 16-bit identifier incremented on a network disconnect that indicates to the server the client's **Channel** change.

Session.EncryptData: A Boolean that, if set, indicates that all messages for this session MUST be encrypted.

Session.EncryptionKey: A 128-bit key used for encrypting the messages sent by the client.

Session.DecryptionKey: A 128-bit key used for decrypting the messages received from the server.

Session.SigningKey: A 128-bit key used for signing the SMB2 messages.

Session.ApplicationKey: A 128-bit key, for the authenticated context, that is queried by the higher-layer applications.

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.

If the client implements the SMB 3.0 dialect, the client MUST also implement the following:

TreeConnect.IsCASHare: A Boolean that, if set, indicates that the tree connect was established on a continuously available share.

TreeConnect.EncryptData: A Boolean that, if set, indicates that the server requires encrypted messages for accessing the share associated with this tree connect.

TreeConnect.IsScaleoutShare: A Boolean that, if set, indicates that the tree connect was established on a share that has the SMB2_SHARE_CAP_SCALEOUT capability set.

3.2.1.5 Per Open File

If the client implements the SMB 2.1 or SMB 3.0 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 can choose to cache writes to the file. The client can 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 can 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 can choose to keep open handles to the file even after the application that opened the file has closed its handles or has ended.

If the client implements the SMB 3.0 dialect, the client MUST implement the following:

- **File.LeaseEpoch:** A sequence number stored by the client to track lease state changes.

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.OpslockLevel: 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.

If the client implements the SMB 3.0 dialect, the client MUST implement the following:

Open.CreateGuid: A unique identifier which identifies the **Open**.

Open.IsPersistent: A Boolean that indicates whether this open is persistent.

Open.DesiredAccess: The access mode requested by the client while opening the file, in the format specified in section [2.2.13.1](#).

Open.ShareMode: The sharing mode requested by the client while opening the file, in the format specified in section [2.2.13](#).

Open.CreateOptions: The create options requested by the client while opening the file, in the format specified in section [2.2.13](#).

Open.FileAttributes: The file attributes used by the client for opening the file, in the format specified in section [2.2.13](#).

Open.CreateDisposition: The create disposition requested by the client for opening the file, in the format specified in section [2.2.13](#).

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.

3.2.1.8 Per Channel

If the client implements SMB 3.0 dialect, the client MUST implement the following:

Channel.SigningKey: 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.

Channel.Connection: A reference to the connection on which this channel was established.

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[70](#) 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[71](#) time-out.

3.2.3 Initialization

ConnectionTable: MUST be set to an empty table.

GlobalFileTable: If implemented, MUST be set to an empty table.

ClientGuid: If implemented, MUST be set to a newly generated GUID.

If the client implements SMB 3.0 dialect:

EncryptionAlgorithmList: MUST be initialized with a list of the encryption algorithms supported by the client.[72](#)

MaxDialect: MUST be set to the highest SMB2 dialect that the client implements.

RequireSecureNegotiate: MUST be set based on the local configuration policy.[73](#)

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.

If **Connection.Dialect** is equal to "3.000" and if **Connection.SupportsMultiChannel** or **Connection.SupportsPersistentHandles** is TRUE, the client MUST set **ChannelSequence** in the SMB2 header to **Session.ChannelSequence**.

3.2.4.1.1 Signing the Message

The client MUST sign the message under the following conditions:

- If the request message being sent contains a nonzero **SessionId** and a zero **TreeId** in the [SMB2 header](#) field, and the session identified by **SessionId** has **Session.SigningRequired** equal to TRUE.
- If the request message being sent contains a nonzero **SessionId** and a nonzero **TreeId** in the SMB2 header field, and the session identified by **SessionId** has **Session.SigningRequired** equal to TRUE and the tree connection identified by **TreeId** has **TreeConnect.EncryptData** equal to FALSE.

If **Session.SigningRequired** is FALSE, the client MAY [<74>](#) sign the request.

If the client implements the SMB 3.0 dialect, and if the request is for session set up, the client MUST use **Session.SigningKey**, and for all other requests the client MUST provide **Channel.SigningKey** by looking up the **Channel** in **Session.ChannelList**, where the connection matches the **Channel.Connection**. Otherwise, the client MUST use **Session.SessionKey** for signing the request. The client provides the key for signing, the length of the request, and the request itself, and calculates the signature as specified in section [3.1.4.1](#). If the client signs the request, it MUST set the SMB2_FLAGS_SIGNED bit in the **Flags** field of the SMB2 header.

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[<75>](#) 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. [<76>](#)

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[<77>](#) 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 SHOULD[<78>](#) 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 MUST 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 can 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[<79>](#) 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[<80>](#) 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.[<81>](#)

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 can 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 a file and close it. This form of compounding MUST NOT be used in combination with compounding unrelated requests within a single send.[<82>](#)

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 0xFFFFFFFFFFFFFF, 0xFFFFFFFF, and { 0xFFFFFFFFFFFFFF, 0xFFFFFFFFFFFFFF }.

3.2.4.1.5 Sending Multi-Credit Requests

A client that implements the SMB 2.1 or SMB 3.0 dialect MAY[<83>](#) 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 or SMB 3.0 dialect and **Connection.SupportsMultiCredit** is FALSE, **CreditCharge** SHOULD[<84>](#) be set to 0 and the payload size of a request or the anticipated response MUST be a maximum of 64 kilobytes.

Otherwise, 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](#).

3.2.4.1.7 Selecting a Channel

If the client implements the SMB 3.0 dialect and if the request being sent is not SMB2_NEGOTIATE or SMB2_SESSION_SETUP, the client MUST choose a channel, to be used for sending the request, from **Session.ChannelList** in an implementation-specific manner[`<85>`](#).

3.2.4.1.8 Encrypting the Message

If the client implements the SMB 3.0 dialect, the client MUST encrypt the message before sending, if any of the following conditions is satisfied:

- If **Session.EncryptData** is TRUE and the request being sent is not SMB2 NEGOTIATE.
- If **Session.EncryptData** is FALSE, the request being sent is not SMB2 NEGOTIATE or SMB2 SESSION_SETUP or SMB2 TREE_CONNECT, and **TreeConnect.EncryptData** is TRUE.

When sending an encrypted message, the client MUST construct the SMB2 TRANSFORM_HEADER specified in section [2.2.41](#) as follows:

- **Nonce** is set to a newly generated implementation-specific value that is never reused for all encrypted messages within the session.
- **OriginalMessageSize** is set to the size of the SMB2 message being sent.
- **SessionId** is set to **Session.SessionId**.
- **EncryptionAlgorithm** is set to a value specified in section [2.2.41](#), corresponding to the encryption algorithm chosen from **EncryptionAlgorithmList** using a local configuration policy.[`<86>`](#)
- **Signature** field is set to a value generated using the using the AES-CCM algorithm as specified in [\[RFC5084\]](#) with the following inputs:
 - The size of the SMB2 TRANSFORM_HEADER excluding the **ProtocolId** and **Signature** fields, as the optional authenticated data length

- The SMB2 TRANSFORM_HEADER excluding the **ProtocolId** and **Signature** fields, as the buffer to sign as the optional authenticated data
- The SMB2 message including the header and the payload, as the data to be signed
- **Session.EncryptionKey** as the key to be used for signing

The client MUST encrypt the SMB2 message using the encryption algorithm chosen above and using **Session.EncryptionKey**.

The client MUST append the encrypted SMB2 message to the SMB2 TRANSFORM_HEADER and send it to the server.

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.[<87>](#) 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. [<88>](#)[<89>](#)

- 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 session with the same user credential 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, the client MAY<90> 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](#).

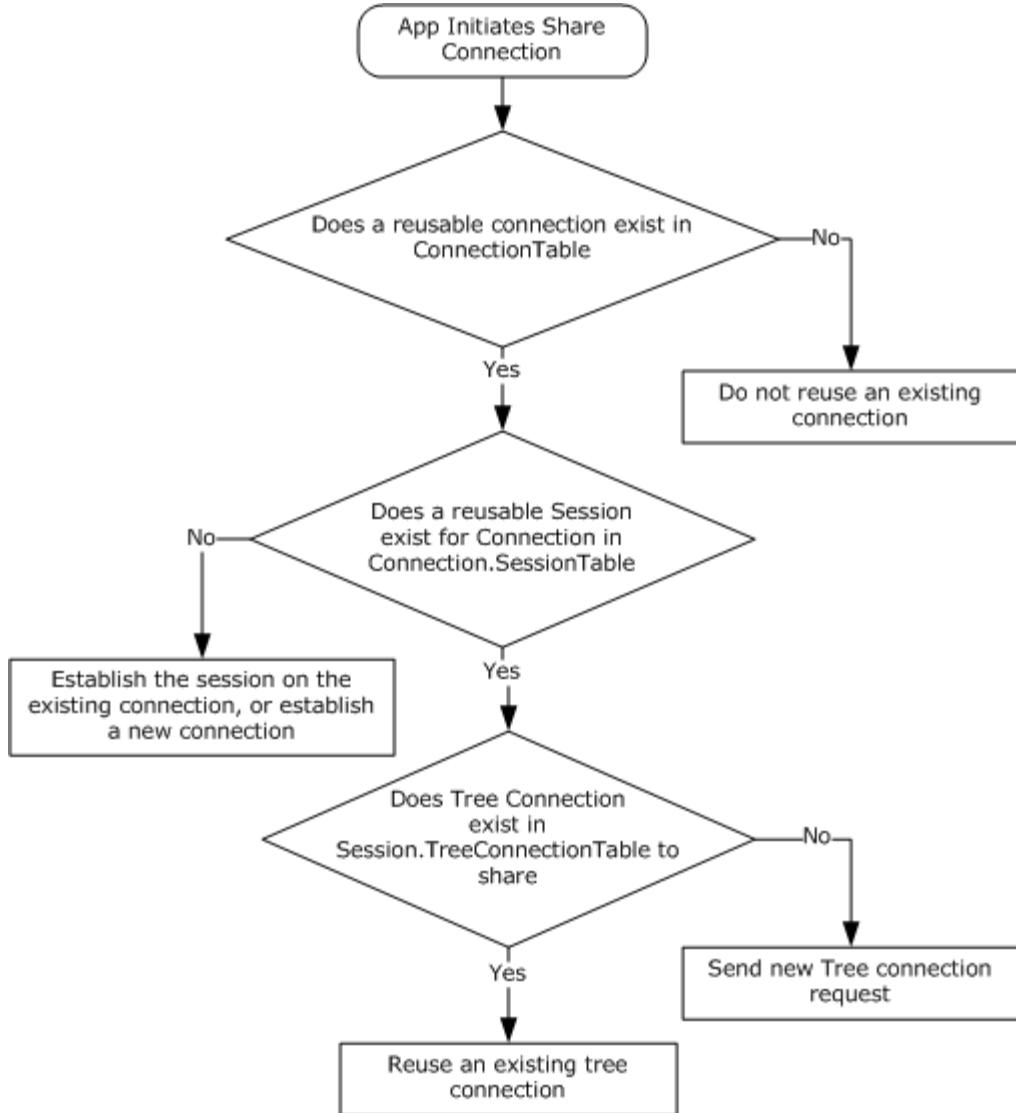


Figure 4: The client MUST follow the steps outlined in this chart.

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-WPO] section 6.1.3, and SHOULD attempt connections to one or more of the returned addresses. The client can 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 can 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<91>. The client can 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.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 [**<92>**](#) 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 implement 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 implement 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 the client implements the SMB 2.002 dialect, it MUST perform the following:

- The client MUST include the dialect string "SMB 2.002" [**<93>**](#) in the list of dialects, along with any other SMB dialects that it implements. The remaining fields in the request MUST be set up as specified in [\[MS-SMB\]](#) section 3.2.4.2.

If the client implements the SMB 2.1 or 3.0 dialects, it MUST perform the following:

- The client MUST include the dialect strings "SMB 2.002" and "SMB 2.????" in the list of dialects, along with any SMB dialects that it implements. The remaining fields 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 the client implements the SMB 3.0 dialect, it MUST do the following:
 - Increment the **DialectCount** by 1.
 - Set the value in the **Dialects[DialectCount-1]** array to 0x0300.
- 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**. The client MUST store the value of the **SecurityMode** field in **Connection.ClientSecurityMode**.
- Set **Capabilities** and **ClientStartTime** to 0.
- If the client implements the SMB 2.1 or SMB 3.0 dialect, **ClientGuid** MUST be set to the global **ClientGuid** value; otherwise it MUST be set to 0.
- If the client implements the SMB 3.0 dialect, the client MUST set the **Capabilities** field as specified in section [2.2.3](#), and store the value of **Capabilities** field in **Connection.ClientCapabilities**.

This request MUST be sent to the server.

3.2.4.2.3 Authenticating the User

To establish a new session, the client MAY[**<94>**](#) 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 <95> 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 **Flags** 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.

If this authentication is for establishing an alternative channel for an existing **Session**, as specified in section [3.2.4.1.7](#), the client MUST also set the following values:

- The **SessionId** field in the SMB2 header MUST be set to the **Session.SessionId** for the new channel being established.
- The SMB2_SESSION_FLAG_BINDING bit MUST be set in the **Flags** field.
- The **PreviousSessionId** field MUST be set to zero.

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](#), or the application or the client itself requests that an existing session be reauthenticated. In either case, the client MUST issue a subsequent session setup request for the **SessionId** of the session being reauthenticated.

The client MAY <96> 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 [<97>](#) 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 **Flags** 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 implements the SMB 2.1 or SMB 3.0 dialect and **Connection.SupportsFileLeasing** is TRUE, 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**,[<98>](#) 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 **Connection.Dialect** is "3.000", **Connection.SupportsDirectoryLeasing** is TRUE, and the file being opened is not the root of the share, the client MUST search the **GlobalFileTable** for the parent directory of the file being opened. The name of the parent directory is obtained by removing the last component of the path used to search the **GlobalFileTable** above. If an entry for the parent directory is not found, a new **File** entry MUST be created for it and added to the **GlobalFileTable** and a **File.LeaseKey**,[<99>](#) 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 **RequestedOlockLevel** 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.[<100>](#)
- 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 do the following:

- If Connection.Dialect is "3.000", the client MUST construct a create context by using the syntax specified in section [2.2.13.2.11](#), with the following values set:
 - **Timeout** MUST be set to an implementation-specific value [`<101>`](#).
 - If **TreeConnect.IsCASHare** is TRUE, the client MUST set the **SMB2_DHANDLE_FLAG_PERSISTENT** bit in the **Flags** field.

- **Reserved** MUST be set to zero.
- **CreateGuid** MUST be set to a newly generated GUID.
- Otherwise, the client MUST construct a create context using the syntax specified in section [2.2.13.2.3](#).
- The client create context 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 as specified in [\[MS-DTYP\]](#) section 2.3.3. 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 or a Directory

To request a lease, in addition to the parameters that are specified in section [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.[<102>](#)

The client MUST fail this request with STATUS_NOT_SUPPORTED in the following cases:

- If **Connection.Dialect** is not equal to "2.100" or "3.000".
- If **Connection.SupportsFileLeasing** is FALSE.
- If **Connection.Dialect** is equal to "2.100" and the open is on a directory.

The client MUST construct an **SMB2 CREATE request** as described in section [3.2.4.3](#), with a **RequestedOplockLevel** of SMB2_OPLOCK_LEVEL_LEASE.

If **Connection.Dialect** is equal to "3.000", the client MUST attach an **SMB2_CREATE_REQUESTLEASE_V2** create context to the request. The create context MUST be formatted as described in section [2.2.13.2.10](#) with the following values:

- **LeaseKey** obtained from **File.LeaseKey** of the file or directory being opened.
- The client MUST search the **GlobalFileTable** for the parent directory of the file being opened. (The name of the parent directory is obtained by removing the last component of the path.) If any entry is found, **ParentLeaseKey** is obtained from **File.LeaseKey** of that entry and **SMB2LEASE_FLAG_PARENTLEASEKEYSET** bit MUST be set in the Flags field.

- **LeaseState** value provided by the application. If the filename to be opened, followed by a ":" colon character and a stream name, indicates a named stream as defined in [MS-FSCC] section 2.1.5, the client SHOULD clear the SMB2_LEASE_HANDLE_CACHING bit in the **LeaseState** field.
- **Epoch** SHOULD be set to 0.

If **Connection.Dialect** is equal to "2.100", the client MUST attach an SMB2_CREATE_REQUESTLEASE 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.

3.2.4.3.11 Application Supplies its Identifier

If **Connection.Dialect** is equal to "3.000", to associate a create or open with an application-supplied identifier, the client MUST construct an SMB2_CREATE_APP_INSTANCE_ID create context by using the syntax specified in section [2.2.13.2.13](#). 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 **RequestedOlockLevel** field is set to **Open.OlockLevel**.
- 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](#).
- If **Connection.Dialect** is "3.000", the client MUST set the following:
 - The client sets the **DesiredAccess** field to **Open.DesiredAccess**.
 - The client sets the **FileAttributes** field to **Open.FileAttributes**.
 - The client sets the **ShareAccess** field to **Open.ShareAccess**.
 - The client sets the **CreateDisposition** field to **Open.CreateDisposition**.
 - The client sets the **CreateOptions** field to **Open.CreateOptions**.
- If **Connection.Dialect** is "2.100", 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 **Connection.Dialect** is "3.000", an [SMB2_CREATE_DURABLE_HANDLE_RECONNECT_V2](#) create context is constructed according to the syntax specified in section [2.2.13.2.12](#). The **FileId** value is set to **Open.FileId**, **CreateGuid** is set to **Open.CreateGuid**, and the create context is appended to the create request. If **Open.Ispersistent** is TRUE, the client MUST set **SMB2_DHANDLE_FLAG_PERSISTENT** bit in the **Flags** field.
- If **Connection.Dialect** is "2.100" or "3.000", and the original open was performed by using a lease as described in section [3.2.4.3.8](#), as indicated by **Open.OlockLevel** set to **SMB2_OPLOCK_LEVEL_LEASE**, it MUST also implement the following:
 - The client MUST re-request the lease as described in section [3.2.4.3.8](#), and the **LeaseState** field MUST be set to **File.LeaseState** of the file being opened.

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.[<103>](#)

If a client requests reading from a file, **Connection.Dialect** is not "2.002", and if **Connection.SupportsMultiCredit** is TRUE, the **CreditCharge** field in the SMB2 header MUST be set to $(1 + (\text{Length} - 1) / 65536)$.

If the **Connection** is established in RDMA mode and the size of any single operation exceeds an implementation-specific threshold [<104>](#), then the interface in [\[MS-SMBD\]](#) section 3.1.4.3 Register Buffer MUST be used to register the buffer on the **Connection** with write permissions, which will receive the data to be read. The returned list of **SMB_DIRECT_BUFFER_DESCRIPTOR_V1** structures MUST be stored in **Request.BufferDescriptorList**. The following fields of the request MUST be initialized as follows:

- The **Channel** field of the request MUST be set to **SMB2_CHANNEL_RDMA_V1**.
- The returned list of **SMB_DIRECT_BUFFER_DESCRIPTOR_1** structures MUST be appended to the SMB2 header.
- The **ReadChannelInfoOffset** MUST be set to the offset of the appended list from the beginning of the SMB2 header.
- The **ReadChannelInfoLength** MUST be set to the length of the appended list.

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.

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 [<105>](#) send these separate writes in any order.

If the Connection is not established in RDMA mode or if the size of the operation is less than or equal to an implementation-specific threshold [<106>](#), then

- 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.[<107>](#)

Otherwise, the interface in [\[MS-SMBD\]](#) section 3.1.4.3 Register Buffer MUST be used to register the buffer on the **Connection** with read permissions, which will supply the data to be written. The returned list of SMB_DIRECT_BUFFER_DESCRIPTOR_V1 structures MUST be stored in **Request.BufferDescriptorList**. The following fields of the request MUST be initialized as follows:

- The **Channel** field of the request MUST be set to SMB2_CHANNEL_RDMA_V1.
- The returned list of SMB_DIRECT_BUFFER_DESCRIPTOR_1 structures MUST be appended to the SMB2 header.
- The **WriteChannelInfoOffset** MUST be set to the offset of the appended list from the beginning of the SMB2 header.

- The **WriteChannelInfoLength** MUST be set to the length of the appended list.

If a client requests writing to a file, **Connection.Dialect** is not "2.002", 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.

The application can also 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, as specified in section [3.3.5.20](#).
- 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.[<108>](#)
- 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 SHOULD 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[]**.[<109>](#)
- 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 [<110>](#) 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 applied, 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 [<111>](#) 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, 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.[<112>](#)
 - 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 the offset, in bytes, 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 [\[MS-FSCC\]](#) 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**.
- The **Flags** field MUST be set to a combination of zero or more of the following bit values, as specified in section [2.2.37](#):
 - SMB2_RESTART_SCANS if the application requested that the enumeration be restarted.
 - SMB2_REOPEN if the application requested that the enumeration be restarted and indicated that the file specifier has changed [`<113>`](#).
 - SMB2_RETURN_SINGLE_ENTRY if the application requested that only a single entry be returned.

If **Connection.Dialect** is not "2.002" 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 the SMB2_LOCKFLAG_SHARED_LOCK bit in the **Flags** field.
 - If the lock is to be acquired exclusively, the client sets the SMB2_LOCKFLAG_EXCLUSIVE_LOCK bit in the **Flags** field.
 - If the lock is to fail immediately if the range is already locked, the client sets the SMB2_LOCKFLAG_FAIL_IMMEDIATELY bit in the **Flags** field. If the **Locks[]** array has more than one element, the client MUST set SMB2_LOCKFLAG_FAIL_IMMEDIATELY.

If any of the Booleans **Open.ResilientHandle**, **Open.IsPersistent**, or **Connection.SupportsMultiChannel** is TRUE, the client MUST do the following:

- 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** be set to **BucketIndex** +1.
- Set **Open.OperationBuckets[BucketIndex].Free** to 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** $\ll 4$) + **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 $\ll 114$ be set to 0.
- The **InputCount** field is set to 0.
- The **OutputOffset** field SHOULD $\ll 115$ 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:

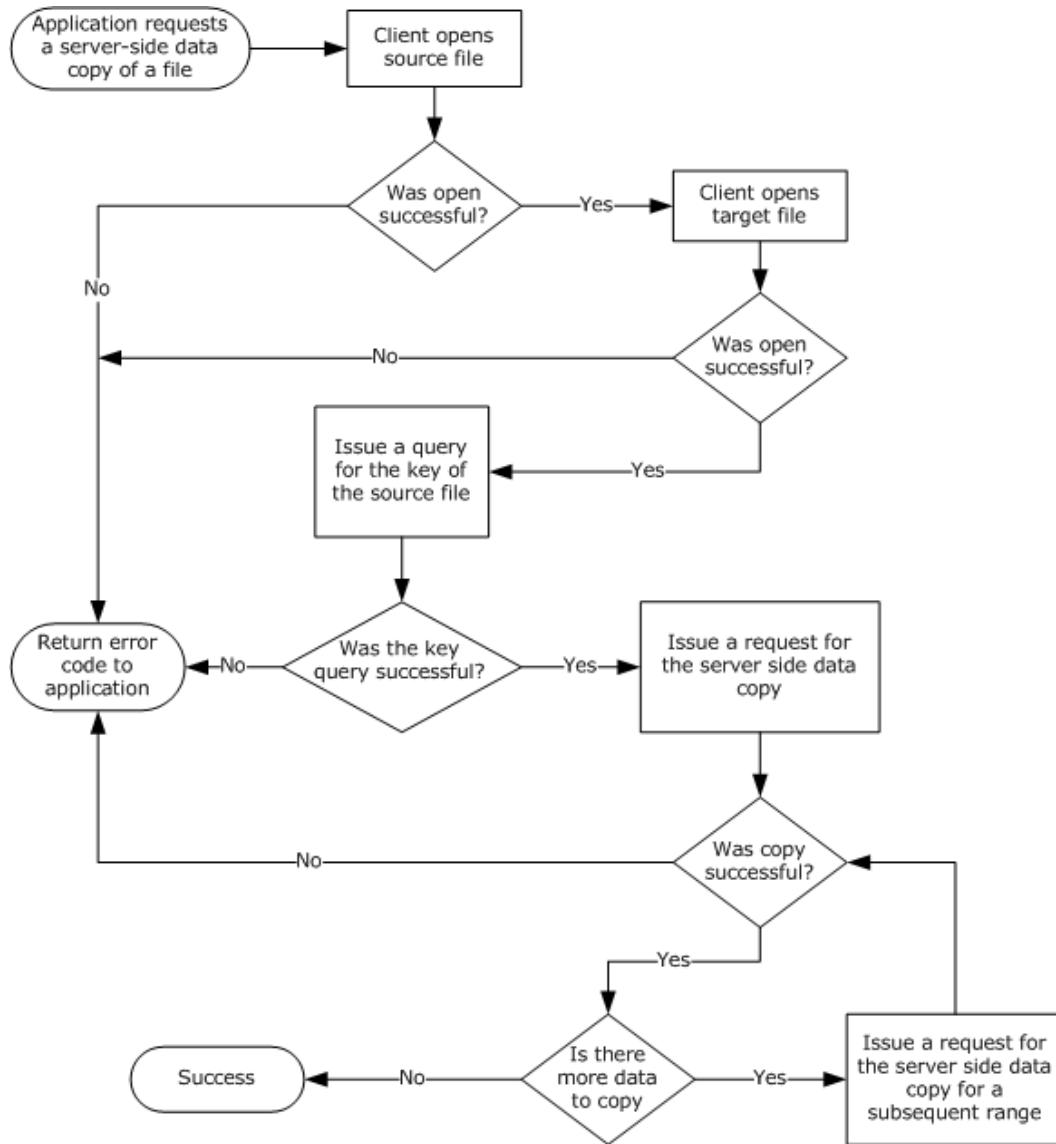


Figure 5: Application requesting a server-side data copy

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 [`<116>`](#) be set to 0.
- The **InputCount** field is set to 0.
- The **OutputOffset** field SHOULD [`<117>`](#) 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.[<118>](#)
- 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[<119>](#) 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.
- **SiteName**: The name of the site to which the client belongs (optional).

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:

- If the **SiteName** is not provided, the **CtlCode** field is set to the operation code to FSCTL_DFS_GET_REFERRALS. Otherwise, the **CtlCode** is set to FSCTL_DFS_GET_REFERRALS_EX.
- The **FileId** field is set to { 0xFFFFFFFFFFFFFF, 0xFFFFFFFFFFFFFF }.
- The **InputOffset** field is set to the offset to the **Buffer[]**, in bytes, from the beginning of the SMB2 header.
- If **SiteName** is not provided, **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.
- If **SiteName** is provided, REQ_GET_DFS_REFERRAL_EX structure, as specified in [\[MS-DFSC\]](#) section 2.2.3, 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 **RequestFlags** is set to 0x001 if the packet contains the **SiteName** of the client requesting referrals.
 - The **RequestDataLength** is set to length of the **requestData** accompanying the referral request.
 - The **requestData** consists of:
 - The **RequestFileNameLength** is set to length of the path for which the referral information is being queried.
 - The **RequestFileName** is set to path for which the referral information is being queried.
 - The **SiteNameLength** is set to length of the **SiteName**.
 - The **SiteName** field is set to **SiteName** supplied by the calling application.
- The **InputCount** field is set to the size of the **Buffer** field.
- The **OutputOffset** field SHOULD [<120>](#) 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 [`<121>`](#) 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 [`<122>`](#) be set to 0.
- The **InputCount** field is set to 0.
- The **OutputOffset** field SHOULD [`<123>`](#) 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 [`<124>`](#) 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 [`<125>`](#) 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 or 3.0 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 **Open.Connection.Dialect** is "2.002", 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 [<126>](#) 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 **Open.Connection.Dialect** is equal to "2.002", 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 [<127>](#) 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 { 0xFFFFFFFFFFFFFF, 0xFFFFFFFFFFFFFF }.
- 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.

3.2.4.20.10 Application Requests Querying Server's Network Interfaces

This optional interface is applicable only for the SMB 3.0 dialect.

The application provides:

- A handle to the **TreeConnect**.

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.

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_QUERY_NETWORK_INTERFACE_INFO.
- The **FileId** field is set to { 0xFFFFFFFFFFFFFF, 0xFFFFFFFFFFFFFF }.
- The **MaxInputResponse** field is set to 0.
- The **MaxOutputResponse** field is set to an implementation-specific [**<128>**](#) value.

- SMB2_0_IOCTL_IS_FSCTL is set to TRUE in the **Flags** field.
 - The **InputOffset** field 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.

3.2.4.21 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 [`<129>`](#) 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 [`<130>`](#) 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.

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 NULL, 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 do the following:

If **Connection.Dialect** is "3.000", the client MUST return **Open.TreeConnect.Session.ApplicationKey**. Otherwise, the client MUST return **Open.TreeConnect.Session.SessionKey**.

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 determine the total number of opens on the **TreeConnect** by enumerating the **Opens** in **TreeConnect.Session.Connection.OpenTable** and counting those **Opens** where

Open.TreeConnect matches the **TreeConnect**. The resulting count is returned to the calling application.

3.2.4.27 Application Notifies Offline Status of a Server

This optional interface is applicable only for the SMB 3.0 dialect. The application provides the following:

- **ServerName:** The name of the server which became unavailable.

For each **Connection** in the **ConnectionTable** where **Connection.ServerName** matches **ServerName**, the client MUST determine if any **TreeConnect** exists in the **Session.TreeConnectTable** with **TreeConnect.IsScaleoutShare** set to TRUE.

If a tree connect entry is found, the client MUST do the following:

- Disconnect the connection by performing the steps as specified in section [3.2.7.1](#).
- Invoke the event as specified in section [3.2.4.28](#) with **ServerName** set to the caller-supplied **ServerName**.

If no tree connect entry is found, the client MUST disconnect the connection by performing the steps as specified in section [3.2.7.1](#).

3.2.4.28 Application Notifies Online Status of a Server

This optional interface is applicable only for the SMB 3.0 dialect. The application provides the following:

- **ServerName:** The name of the server which became available.

For each **Open** in the **GlobalFileTable**, where **Open.Connection** is NULL and the server name identified from **Open.FileName** matches **ServerName**, the client MUST re-establish the durable open as specified in section [3.2.4.3](#).

3.2.4.29 Application Requests Moving to a Server Instance

This optional interface is applicable only for SMB 3.0 dialect. The application provides the following:

- **ServerName:** The name of the server.
- **NewServerAddress:** The IPv4 or IPv6 address of the server which the client is required to move to.

For each **Connection** in the **ConnectionTable** where **Connection.ServerName** matches **ServerName**, the client MUST disconnect the connection by performing the steps as specified in section [3.2.7.1](#).

For each **Open** in the **GlobalFileTable**, where **Open.Connection** is NULL and the server name identified from **Open.FileName** matches **ServerName**, the client MUST re-establish the durable open as specified in section [3.2.4.4](#), and by using **NewServerAddress** as the **TransportIdentifier** for the rules specified in section [3.2.4.2](#).

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 Decrypting the Message

This section is applicable for only the SMB 3.0 dialect.

If the **ProtocolId** in the header of the received message is 0x424d53FD, the client MUST perform the following:

- If the size of the message received from the server is not greater than the size of SMB2_TRANSFORM_HEADER as specified in section [2.2.41](#), the client MUST discard the message.
- If the **EncryptionAlgorithm** in the SMB2 TRANSFORM_HEADER is not a valid algorithm listed in **EncryptionAlgorithmList**, the client MUST discard the message.
- The client MUST look up the session in the **Connection.SessionTable** using the **SessionId** in the SMB2 TRANSFORM_HEADER of the response. If the session is not found, the response MUST be discarded as invalid.
- The client MUST decrypt the message using **Session.DecryptionKey** and the algorithm specified by the **EncryptionAlgorithm** in TRANSFORM_HEADER. The client passes in the TRANSFORM_HEADER, excluding the **Signature** and **ProtocolId** fields, and the encrypted SMB2 message as the Optional Authenticated Data input for AES-CCM. If decryption succeeds, the client compares the signature in the transform header with the signature returned by the decryption algorithm. If signature verification succeeds, the client MUST then continue processing the decrypted packet, as specified in subsequent sections.

3.2.5.1.2 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 0xFFFFFFFFFFFFFF, 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.

3.2.5.1.3 Verifying the Signature

If the client implements the SMB 3.0 dialect and if the decryption in section [3.2.5.1.1](#) succeeds, the client MUST skip the processing in this section.

If the **MessageId** is 0xFFFFFFFFFFFFFF, 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 **Connection.Dialect** is "3.000", and the received message is an SMB2 SESSION_SETUP Response, the client MUST verify the signature of the message as specified in section [3.1.5.1](#), using **Session.SigningKey** as the signing key, and passing the response message.

If **Connection.Dialect** is "3.000", and the received message is not an SMB2 SESSION_SETUP Response, the client MUST look up the **Channel** in **Session.ChannelList**, where the **Channel.Connection** matches the connection on which this message is received. The client MUST verify the signature of the message as specified in section [3.1.5.1](#), using **Channel.SigningKey** as the signing key, and passing the response message.

Otherwise, the client MUST verify the signature of the message as specified in section [3.1.5.1](#), using **Session.SessionKey** as the signing 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.SigningRequired** 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.SigningRequired** is FALSE, the client continues processing on the packet, as specified in subsequent sections. [<131>](#)

3.2.5.1.4 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.

3.2.5.1.5 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.

3.2.5.1.6 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](#).

3.2.5.1.7 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 [`<132>`](#) also disconnect the connection.

3.2.5.1.8 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 [`<133>`](#) fail the application request with an implementation-specific error that indicates an invalid network response was received.

3.2.5.1.9 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, regardless of any compounding used when sending the requests.

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**.[`<134>`](#) 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 [3.2.4.2.2.2](#) with the only exception that the client MUST allocate sequence number 1 from **Connection.SequenceWindow**, and MUST set **MessageId** field of the SMB2 header to 1. Otherwise, the client MUST proceed as follows.

If the client implements SMB 2.1 or SMB 3.0, 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.SupportsFileLeasing** 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.

If **Connection.Dialect** is "3.000", the client MUST perform the following:

- If SMB2_GLOBAL_CAP_DIRECTORY_LEASING is set in the **Capabilities** field of the SMB2 NEGOTIATE Response, the client MUST set **Connection.SupportsDirectoryLeasing** to TRUE. Otherwise, it MUST be set to FALSE.
- If SMB2_GLOBAL_CAP_MULTI_CHANNEL is set in the **Capabilities** field of the SMB2 NEGOTIATE Response, the client MUST set **Connection.SupportsMultiChannel** to TRUE. Otherwise, it MUST be set to FALSE.
- If SMB2_GLOBAL_CAP_PERSISTENT_HANDLES is set in the **Capabilities** field of the SMB2 NEGOTIATE Response, the client SHOULD invoke the event as specified in [\[MS-SWN\]](#) section 3.2.4.1 by providing **Connection.ServerName** as *Netname* parameter.
- If SMB2_GLOBAL_CAP_ENCRYPTION is set in the **Capabilities** field of the SMB2 NEGOTIATE Response, the client MUST set **Connection.SupportsEncryption** to TRUE. Otherwise, it MUST be set to FALSE.
- **Connection.ServerCapabilities** MUST be set to the **Capabilities** field of the SMB2 NEGOTIATE Response.
- **Connection.ServerSecurityMode** MUST be set to the **SecurityMode** field of the SMB2 NEGOTIATE Response.

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** by 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, as specified in section [3.2.5.3.1](#).

If a session is located:

- If **Session.Connection** matches the connection on which this response is received, this response MUST be handled as a reauthentication, as specified in section [3.2.5.3.2](#).
- If **Connection.Dialect** is "3.000", and if there is no **Channel** in **Session.ChannelList** where the **Channel.Connection** matches the connection on which this response is received, this response MUST be handled as a session binding, as specified in section [3.2.5.3.3](#).

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. If **Connection.Dialect** is "3.000", the client MUST generate **Session.SigningKey**, as specified in section [3.1.4.2](#), and passing the following inputs:
 - **Session.SessionKey** as the key derivation key.
 - The case-sensitive ASCII string "SMB2AESCMAC" as the label.
 - The label buffer size in bytes, including the terminating null character. The size of "SMB2AESCMAC" is 12.
 - The case-sensitive ASCII string "SmbSign" as context for the algorithm.
 - The context buffer size in bytes, including the terminating null character. The size of "SmbSign" is 8.
- If **Connection.Dialect** is "3.000", **Session.ApplicationKey** MUST be generated as specified in section [3.1.4.2](#), and passing the following inputs:
 - **Session.SessionKey** as the key derivation key.

- The case-sensitive ASCII string "SMB2APP" as the label.
- The label buffer size in bytes, including the terminating null character. The size of "SMB2APP" is 8.
- The case-sensitive ASCII string "SmbRpc" as context for the algorithm.
- The context buffer size in bytes, including the terminating null character. The size of "SmbRpc" is 7.
- **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.SigningRequired** MUST be set to TRUE, otherwise **Session.SigningRequired** MUST be set to FALSE.
- If the security subsystem indicates that the session was established by an anonymous user, **Session.SigningRequired** 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.SigningRequired** is TRUE, this indicates a SESSION_SETUP failure and the connection MUST be terminated. If the SMB2_SESSION_FLAG_IS_GUEST bit is set in the **SessionFlags** field of the SMB2 SESSION_SETUP Response AND if [RequireMessageSigning](#) is FALSE, **Session.SigningRequired** MUST be set to FALSE.
- If **Connection.Dialect** is "3.000" and if the SMB2_SESSION_FLAG_ENCRYPT_DATA bit is set in the **SessionFlags** field of the SMB2 SESSION_SETUP Response, and **Connection.SupportsEncryption** is TRUE, **Session.EncryptData** MUST be set to TRUE, and **Session.SigningRequired** MUST be set to FALSE. The client MUST generate **Session.EncryptionKey** and **Session.DecryptionKey**, as specified in section [3.1.4.2](#) and passing the following inputs:
 - **Session.SessionKey** as the key derivation key.
 - The case-sensitive ASCII string "SMB2AESCCM" as the label.
 - The label buffer length in bytes, including the terminating null character. The size of "SMB2AESCCM" is 11.
 - The case-sensitive ASCII string as key derivation context. For generating the encryption key, this MUST be "ServerIn "; note the blank space at the end. For generating the decryption key, this MUST be "ServerOut".
 - The context buffer size in bytes, including the terminating null character. For generating both the encryption key and decryption key, the string size is 10.
- **Session.OpenTable** MUST be set to an empty table.

If **Connection.Dialect** is "3.000", the client MUST insert a new channel entry in **Session.ChannelList**.

- **Channel.SigningKey**: MUST be set to **Session.SigningKey**.
- **Channel.Connection**: MUST be set to the **Session.Connection**.

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 **Flags** 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.

If **Connection.Dialect** is "3.000", and the request is for establishing a new channel, the client MUST also implement the following:

- The **SessionId** field in the SMB2 header MUST be set to the **Session.SessionId** for the new channel being established. The SMB2_SESSION_FLAG_BINDING bit MUST be set in the **Flags** field.
- The request MUST be signed as specified in section [3.2.4.1.1](#).

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 **Flags** 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.

3.2.5.3.3 Handling Session Binding

The processing in this section is only applicable to a client that implements the SMB 3.0 dialect.

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 caller that initiated the session binding 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, specified by the **SecurityBufferOffset** and **SecurityBufferLength** fields. 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 caller that initiated the session binding 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, session binding is complete. The client MUST insert a new **Channel** entry in **Session.ChannelList** with the following values set:

- **Channel.SigningKey**: MUST be set to a new signing key generated as specified in section [3.1.4.2](#), and passing the following inputs:
 - The value queried from the GSS protocol, as above, as the key derivation key.
 - The case-sensitive ASCII string "SMB2AESCMAC" as the label.
 - The label buffer size in bytes, including the terminating null character. The size of "SMB2AESCMAC" is 12.
 - The case-sensitive ASCII string "SmbSign" as context for the algorithm.
 - The context buffer size in bytes, including the terminating null character. The size of "SmbSign" is 8.
- **Channel.Connection**: MUST be set to the **Connection** on which this response is received.

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 Flags field to zero.
- 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.
 - The **SessionId** field in the SMB2 header MUST be set to the **Session.SessionId** for the new channel being established.

- The SMB2_SESSION_FLAG_BINDING bit MUST be set in the **Flags** field.

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**.

If **Connection.Dialect** is "3.000", the client MUST locate the **Session** in **Session.ChannelList** and remove all entries.

For each connection in **ConnectionTable**, the associated session object MUST be removed.

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.IsCASHare** MUST be set to TRUE, if the SMB2_SHARE_CAP_CONTINUOUS_AVAILABILITY 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.
- If Connection.Dialect is "3.000", **Connection.SupportsEncryption** is TRUE, and if the SMB2_SHAREFLAG_ENCRYPT_DATA bit is set in the **ShareFlags** field of the response, the client MUST do the following:
 - **TreeConnect.EncryptData** MUST be set to TRUE.
 - If **Session.EncryptData** is FALSE, the client MUST then generate an encryption key, a decryption key as specified in section [3.1.4.2](#), by providing the following inputs and store them in **Session.EncryptionKey** and **Session.DecryptionKey**:
 - **Session.SessionKey** as the key derivation key.

- The case-sensitive ASCII string "SMB2AESCCM" as the label.
- The label buffer length in bytes, including the terminating null character. The size of "SMB2AESCCM" is 11.
- The case-sensitive ASCII string as key derivation context. For generating the encryption key, this MUST be "ServerIn "; note the blank space at the end. For generating the decryption key, this MUST be "ServerOut".
- The context buffer size in bytes, including the terminating null character. For generating both the encryption key and decryption key, the string size is 10.

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

If **Connection.Dialect** is "3.000" and the **Capabilities** field in the response includes SMB2_SHARE_CAP_CLUSTER bit, the client SHOULD invoke the event as specified in [\[MS-SWN\]](#) section 3.2.4.1 by providing **Connection.ServerName** as *Netname* parameter.

If **Connection.Dialect** is "3.000" and the **Capabilities** field in the response includes the SMB2_SHARE_CAP_SCALEOUT bit, the client MUST set **TreeConnect.IsScaleoutShare** to TRUE.

If **MaxDialect** is "3.000", and **RequireSecureNegotiate** is TRUE, the client MUST validate the SMB2 NEGOTIATE messages originally sent on this connection by sending a signed VALIDATE_NEGOTIATE_INFO request as specified in section [2.2.31.4](#). The client MUST issue an SMB2 IOCTL Request as follows:

- 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_VALIDATE_NEGOTIATE_INFO.
 - The **FileId** field is set to { 0xFFFFFFFFFFFFFF, 0xFFFFFFFFFFFFFF }.

- The **InputOffset** field is set to the offset of the **Buffer[]**, in bytes, from the beginning of the SMB2 header.
- The **InputCount** field is set to the size, in bytes, of the VALIDATE_NEGOTIATE_INFO request structure that is constructed following the syntax specified in section [2.2.31.4](#).
- The VALIDATE_NEGOTIATE_INFO request structure is constructed as follows and copied into the request at **InputOffset** bytes from the beginning of the SMB2 header:
 - **Capabilities** is set to **Connection.ClientCapabilities**.
 - **Guid** is set to the global **ClientGuid** value.
 - **SecurityMode** is set to **Connection.ClientSecurityMode**.
 - 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 the client implements the SMB 3.0 dialect, it MUST do the following:
 - Increment the **DialectCount** by 1.
 - Set the value in the **Dialects[DialectCount-1]** array to 0x0300.
- The **OutputOffset** field offset to the **Buffer[]**, in bytes, from the beginning of the SMB2 header.
- The **OutputCount** field is set to 0.
- The **MaxInputResponse** field is set to 0.
- The **MaxOutputResponse** field is set to the size of the VALIDATE_NEGOTIATE_INFO response structure as defined in section [2.2.32.6](#).
- The value of the **Flags** field is set to SMB2_0_IOCTL_IS_FSCTL.
- The request MUST be signed as specified in section [3.1.4.1](#), MUST be sent to the server, and the response from the server MUST be handled as specified in section [3.2.5.14.12](#).

If **Connection.Dialect** is "3.000" and **Connection.SupportsMultiChannel** is TRUE, the client MUST perform the following:

- The client MUST verify if the session requires additional channels to the server, in an implementation-specific manner.[<135>](#)
- If the session requires additional channels, the client MUST query the network interfaces on the server, as specified in section [3.2.4.20.10](#), and passing the **TreeConnect**.

- From the list of network interfaces returned by the server, as specified in section [3.2.5.14.10](#), the client MUST select a network interface for establishing a new channel using an implementation-specific criteria.[<136>](#)
- For each server's network interface selected, the client MUST establish a new transport connection to the server, as specified in section [3.2.4.2.1](#).
- The client MUST send SMB2 NEGOTIATE request on the new connection, as specified in section [3.2.4.2.2.2](#).
- If the SMB2 NEGOTIATE request is successful, the client MUST bind the current **Session** to the new connection as specified in section [3.2.4.2.3](#).

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. If **Connection.Dialect** is "3.000" and if **Session.TreeConnectTable** is empty in all sessions in the **Connection.SessionTable** for which **Connection.ServerName** matches the server name, the client SHOULD invoke the event as specified in [\[MS-SWN\]](#) section 3.2.4.3.

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.OlockLevel** MUST be set to the **OlockLevel** 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 **Connection.Dialect** is "3.000", the client MUST set the following:

- Open.DesiredAccess** MUST be set to the **DesiredAccess** field of the request.
- Open.ShareMode** MUST be set to the **ShareAccess** field of the request.
- Open.CreateOptions** MUST be set to the **CreateOptions** field of the request.
- Open.FileAttributesOptions** MUST be set to the **FileAttributes** field of the request.
- Open.CreateDisposition** MUST be set to the **CreateDisposition** field of the request.

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 **Session.OpenTable**. If **Connection.Dialect** is not "2.002" and **Connection.SupportsFileLeasing** 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 **Connection.Dialect** is "3.000" and **Connection.SupportsDirectoryLeasing** is TRUE, the client MUST search the **GlobalFileTable** for the parent directory of the file being opened. The name of the parent directory is obtained by removing the last component of the path used to search the **GlobalFileTable** above. 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.

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_RESPONSELEASE Create Context

If **Connection.Dialect** is not "2.002" and an SMB2_CREATE_RESPONSELEASE create context is present in the [SMB2_CREATE response](#) returned from the server, it MUST do the following:

- If **Connection.SupportsFileLeasing** 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**.

3.2.5.7.5 SMB2_CREATE_RESPONSELEASE_V2 Create Context

If **Connection.Dialect** is "3.000" and an SMB2_CREATE_RESPONSELEASE_V2 create context is present in the SMB2_CREATE response returned from the server, the client MUST locate the **File** entry corresponding to **Open.FileName** in the **GlobalFileTable**.

The client MUST evaluate Δ_{epoch} depending on the lease state change indicated in the table below. The rows in the table represent the lease state currently held by the client (**File.LeaseEpoch**) and the columns represent the **LeaseState** returned in the SMB2_CREATE_RESPONSELEASE_V2 create context.

Δ_{epoch} is a 16-bit signed integer indicating if the Epoch value sent by the server is newer than the current Epoch value held by the client. It is evaluated as follows:

- If the **Epoch** value sent by the server is equal to **File.LeaseEpoch**, then Δ_{epoch} is 0.
- If the **Epoch** value sent by the server is not equal to **File.LeaseEpoch** and **Epoch** value sent by the server minus **File.LeaseEpoch** is less than or equal to 32767, then Δ_{epoch} is greater than 0.

New Lease State	R	RH	RWH	None
None	$\Delta_{epoch}=0$: Invalid $\Delta_{epoch}>0$: Upgrade	$\Delta_{epoch}=0$: Invalid $\Delta_{epoch}>0$: Upgrade	$\Delta_{epoch}=0$: Invalid $\Delta_{epoch}>0$: Upgrade	
R	$\Delta_{epoch}=0$: No change $\Delta_{epoch}>0$: Purge cache	$\Delta_{epoch}=1$: Upgrade $\Delta_{epoch}>1$: Upgrade & purge cache. $\Delta_{epoch}=0$: Invalid.	$\Delta_{epoch}=1$: Upgrade $\Delta_{epoch}>1$: Upgrade & purge cache. $\Delta_{epoch}=0$: Invalid	$\Delta_{epoch}>0$: Purge cache
RH	Invalid	$\Delta_{epoch}=0$: No change $\Delta_{epoch}>0$: Purge cache	$\Delta_{epoch}=1$: Upgrade $\Delta_{epoch}>1$: Upgrade & purge cache. $\Delta_{epoch}=0$: Invalid	
RWH	Invalid	Invalid	$\Delta_{epoch}=0$: No change $\Delta_{epoch} \neq 0$: Invalid	

When Δ_{epoch} indicates an upgrade to a new lease state, the client MUST perform the following:

- Set **File.LeaseState** to the **LeaseState** returned in the create context.
- Set **File.LeaseEpoch** to the **Epoch** returned in the create context.

When Δ_{epoch} indicates Purge cache, the client MUST purge any cached data.

3.2.5.7.6 SMB2_CREATE_DURABLE_HANDLE_RESPONSE_V2 Create Context

If the SMB2_CREATE_DURABLE_HANDLE_RESPONSE_V2 context is present, the client MUST set **Open.Durable** to TRUE. Otherwise, the client MUST set **Open.Durable** to FALSE. If the SMB2_DHANDLE_FLAG_PERSISTENT bit is set in the **Flags** field of the SMB2_CREATE_DURABLE_HANDLE_RESPONSE_V2 context, the client MUST set **Open.IsPersistent** to TRUE, otherwise set to FALSE.

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 **Session.OpenTable**.

If **Connection.Dialect** is not "2.002" and **Connection.SupportsFileLeasing** 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 **Connection.Dialect** is not "2.002" and an SMB2_CREATE_RESPONSELEASE create context is present in the SMB2 CREATE response returned from the server, the client MUST do the following:

- If **Connection.SupportsFileLeasing** 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**.

If **Connection.Dialect** is "3.000" and an SMB2_CREATE_RESPONSELEASE_V2 create context is present in the SMB2 CREATE response returned from the server, the client MUST do the following:

- If **Connection.SupportsDirectoryLeasing** 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**.

If **Connection.Dialect** is "3.000" and the **Connection.SupportsDirectoryLeasing** is TRUE, the client MUST search the **GlobalFileTable** for the parent directory of the file opened. The name of the parent directory is obtained by removing the last component of the path in **Open.FileName**. 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**.

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 **Session.OpenTable** using the **FileId** in the SMB2 header of the response.

If **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, and if there is no entry in **GlobalFileTable** whose name with its last component removed matches **Open.FileName** then the entry for this **File** MUST be deleted from the **GlobalFileTable**, and the **File** object MUST be freed.

If **Connection.Dialect** is "3.000" and **Connection.SupportsDirectoryLeasing** is TRUE, and if the **File** object was freed above, the client MUST scan through the **GlobalFileTable** and remove all **File** objects where **File.OpenTable** is empty and there is no entry in **GlobalFileTable** whose name with its last component removed matches the name of this **File** entry (that is, no child objects exist).

The open object MUST be removed from the **Session.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 **Connection.Dialect** is "3.000", the underlying transport is RDMA, and **Request.BufferDescriptorList** is not empty, then the processing specified in [\[MS-SMBD\]](#) section 3.1.4.4 Deregister Buffer MUST be used to deregister the buffer before returning to the application.

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

If **Connection.Dialect** is "3.000", the underlying transport is RDMA and **Request.BufferDescriptorList** is not empty, then the processing specified in [MS-SMBD] section 3.1.4.4 Deregister Buffer MUST be used to deregister the buffer before returning to the application.

If **Connection.Dialect** is "3.000" and the status code is STATUS_FILE_NOT_AVAILABLE, the client SHOULD^{<137>} replay the write request by looking up the request in **Connection.OutstandingRequests** using the **MessageId** field of the SMB2 header.

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 **Connection.Dialect** is "3.000" and the status code is STATUS_FILE_NOT_AVAILABLE, the client SHOULD^{<138>} replay the IOCTL request by looking up the request in **Connection.OutstandingRequests** using the **MessageId** field of the SMB2 header.

If the **OutputCount** field in an [SMB2 IOCTL Response](#) is 0, the **OutputOffset** field SHOULD^{<139>} 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 **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[<140>](#) 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.

3.2.5.14.11 Handling a Network Interfaces Response

The client MUST return the list of network interfaces received from the server to the calling application.

3.2.5.14.12 Handling a Validate Negotiate Info Response

If the response is not signed or the signature verification in section [3.2.5.1.3](#) does not succeed, the client MUST terminate the **Connection**.

If the **Status** field of the SMB2 header of the response is STATUS_ACCESS_DENIED, the client MUST terminate the **Connection**.

If the **Status** field of the SMB2 header of the response indicates success, the client MUST verify the VALIDATE_NEGOTIATE_INFO response received in the Buffer field of the SMB2 IOCTL Response as follows:

- **Capabilities** MUST be equal to **Connection.ServerCapabilities**.
- **Guid** MUST be equal to **Connection.ServerGuid**.
- **SecurityMode** MUST be equal to **Connection.ServerSecurityMode**.
- **Dialect** MUST be equal to **Connection.Dialect**.

If any of the above verifications fails, the client MUST close all the sessions in **Connection.SessionTable** as specified in section [3.2.4.23](#) and MUST terminate the **Connection**.

Otherwise, the result is successful.

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

If **Connection.Dialect** is "3.000" and the status code is **STATUS_FILE_NOT_AVAILABLE**, the client [SHOULD<141>](#) replay the SetInfo request by looking up the request in **Connection.OutstandingRequests** using the **MessageId** field of the SMB2 header.

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 0xFFFFFFFFFFFFFF, this MUST be processed as an oplock break indication. Otherwise, the client MUST process it as a response to an oplock break acknowledgment.

If **Connection.Dialect** is not "2.002", the client MUST verify:

- If **Connection.SupportsFileLeasing** is TRUE or **Connection.SupportsDirectoryLeasing** is TRUE, the client MUST use the **StructureSize** field in the SMB2 OPLOCK_BREAK notification to differentiate between an oplock break notification and a lease break notification as specified in [2.2.25](#).

3.2.5.19.1 Receiving an Oplock Break Notification

The client MUST locate the open in the **Session.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<142> 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 **Connection.Dialect** is not "2.002", the client MUST verify:

If **Connection.SupportsDirectoryLeasing** is TRUE or **Connection.SupportsFileLeasing** 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 SMB2LEASE_WRITE_CACHING and the new LeaseState does not include SMB2LEASE_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 SMB2LEASE_READ_CACHING and the new LeaseState does not include SMB2LEASE_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](#).
- If **Connection.Dialect** is "3.000" and **File.LeaseState** is equal to the new LeaseState and (**NewEpoch** - **File.LeaseEpoch**) is greater than 1, the client MUST purge any cached data.
- If **Connection.Dialect** is "3.000" and **NewEpoch** is greater than **File.LeaseEpoch**, the client MUST copy the new **LeaseState** into **File.LeaseState**. The client MUST set **File.LeaseEpoch** to **NewEpoch**.
- If **Connection.Dialect** is "2.100", 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_BREAKLEASE_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 [choose any time-out it requires based on local policy, the type of request, and network characteristics.](#)

Clients SHOULD [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 3.2.5.1.5.](#)

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 [return an implementation-specific error to the calling application.](#)

The client MAY [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 in **ConnectionTable**, for each session in **Connection.SessionTable**, if **Session.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, for each **Session** in **Connection.SessionTable**, the client MUST perform the following:

- If **Connection.Dialect** is "3.000", and if the **Session** has more than one channel in **Session.ChannelList**, the client MUST perform the following actions:
 - The channel entry MUST be removed from the **Session.ChannelList**, where **Channel.Connection** matches the disconnected connection.
 - For each outstanding create request in **Connection.OutstandingRequests** containing SMB2_CREATE_DURABLE_HANDLE_REQUEST_V2 context, the client MUST replay the create request on an alternate channel by setting the SMB2_FLAGS_REPLAY_OPERATION bit in the SMB2 header.
 - **Session.ChannelSequence** MUST be incremented by 1.
- Otherwise, the client MUST perform the following actions:
 - For each **Open** in **Session.OpenTable**:
 - If **Connection.Dialect** is not "2.002" and **Connection.SupportsFileLeasing** 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 **Connection.Dialect** is "3.000" and if **Connection.SupportsDirectoryLeasing** is TRUE, and if all opens in **File.OpenTable** are deleted and if there is no entry in the **GlobalFileTable** whose name with its last component removed matches the

Open.FileName, then the entry for the **File** MUST be deleted from the **GlobalFileTable**, and the **File** object MUST be freed.

- Otherwise, 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 **Session.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 **Session.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 **Session.OpenTable** and the **Open** MUST be freed, and the handle generated for the **Open** MUST be invalidated.
- Each **TreeConnect** in **Session.TreeConnectTable** MUST be freed, the handle generated for the **TreeConnect** MUST be invalidated, and the **TreeConnect** entry MUST be removed from **Session.TreeConnectTable**.
- If **Connection.Dialect** is "3.000", the client MUST free the channel and remove the channel entry in **Session.ChannelList**.
- The client MUST free the **Session** and invalidate the session handle.

If **Connection.Dialect** is "3.000" and if **Session.TreeConnectTable** is empty in all sessions in the **Connection.SessionTable** for which **Connection.ServerName** matches the server name, the client SHOULD invoke the event as specified in [\[MS-SWN\]](#) section 3.2.4.3.

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[<147>](#) 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 negotiated dialect is SMB 2.1 or SMB 3.0 and the request is a multi-credit request, it consumes sequence numbers based on the **CreditCharge** field in the SMB2 header, as specified in [3.3.5.2.3](#).

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[<148>](#) grant any number of credits up to that which the client requests, or more if required by the preceding rule.

- The server SHOULD [<149>](#) 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 [<150>](#) 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 the SMB 2.1 or SMB 3.0 dialect 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 SMB2 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 SMB2 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 SMB2 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 SMB2 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 SMB2 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^{[151](#)} 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

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 the SMB 2.1 or SMB 3.0 dialect, 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 the SMB 2.1 or SMB 3.0 dialect 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.

If the server implements the SMB 3.0 dialect, it MUST implement the following:

- **EncryptionAlgorithmList**: A list of strings containing the encryption algorithms supported by the server.
- **EncryptData**: A Boolean that, if set, indicates that this server supports message encryption.
- **RejectUnencryptedAccess**: A Boolean that, if set, indicates that the server will reject any unencrypted messages. This flag is applicable only if **EncryptData** is TRUE or if **Share.EncryptData** (as defined in section [3.3.1.6](#)) is TRUE.

3.3.1.6 Per Share

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 [<152>](#) 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.[<153>](#)
- **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.[<154>](#)
- **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.ForceLevel2Olock**: 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.

If the server implements the SMB 3.0 dialect, it MUST implement the following:

- **Share.CATimeout**: The minimum time, in milliseconds, before closing an unreclaimed persistent handle on a continuously available share.
- **Share.IsCA**: A Boolean that, if set, indicates that the share is continuously available.
- **Share.EncryptData**: A Boolean that, if set, indicates that the server supports encrypted messages for accessing this share.

3.3.1.7 Per Transport Connection

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 [2.2.3](#).
- **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", "3.000" 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-16 IP 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.
- **Connection.SessionTable**: A table of authenticated sessions, as specified in section [3.3.1.8](#), 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.

If the server implements the SMB 2.1 or 3.0 dialect, it MUST implement the following:

- **Connection.ClientGuid**: An identifier for the client machine.

If the server implements the SMB 3.0 dialect, it MUST implement the following:

- **Connection.ServerCapabilities**: The capabilities sent by the server in the SMB2 NEGOTIATE Response on this connection, in a form that MUST follow the syntax as specified in section [2.2.4](#).
- **Connection.ClientSecurityMode**: The security mode sent by the client in the SMB2 NEGOTIATE request on this connection, in a form that MUST follow the syntax as specified in section [2.2.3](#).
- **Connection.ServerSecurityMode**: The security mode received from the server in the SMB2 NEGOTIATE response on this connection, in a form that MUST follow the syntax as specified in section [2.2.4](#).

3.3.1.8 Per Session

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.IsGuest**: A Boolean that, if set, indicates that the session is for a guest 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.SigningRequired**: A Boolean that, if set, indicates that all of the messages for this session MUST be signed.
- **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.

If the server implements the SMB 3.0 dialect, it MUST implement the following:

- **Session.ChannelList**: A list of channels that have been established on this authenticated session, as specified in section [3.3.1.14](#).
- **Session.EncryptData**: A Boolean that, if set, indicates that all of the messages for this session SHOULD be encrypted.
- **Session.EncryptionKey**: A 128-bit key used for encrypting the messages sent by the server.
- **Session.DecryptionKey**: A 128-bit key used for decrypting the messages received from the client.
- **Session.SigningKey**: A 128 bit key used for signing the SMB2 messages.
- **Session.ApplicationKey**: A 128-bit key, for the authenticated context, that is queried by the higher-layer applications.

3.3.1.9 Per Tree Connect

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

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 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_LEVELLEASE.
- **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 or SMB 3.0 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 Unicode string that contains the local path name on the server that the open is performed on.

If the server implements the SMB 2.1 or SMB 3.0 dialect and supports leasing, it MUST implement the following:

- **Open ClientGuid**: An identifier for the client machine that created this open.
- **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.[<155>](#)

If the server implements the SMB 3.0 dialect, it MUST implement the following:

- **Open CreateGuid**: A 16-byte value that associates this open to a create request.
- **Open AppInstanceId**: A 16-byte value that associates this open with a calling application.
- **Open IsPersistent**: A Boolean that indicates whether this open is persistent.
- **Open ChannelSequence**: A 16-bit identifier indicating the client's **Channel** change.
- **Open OutstandingRequestCount**: A numerical value that indicates the number of outstanding requests issued with **ChannelSequence** equal to **Open.ChannelSequence**.
- **Open OutstandingPreRequestCount**: A numerical value that indicates the number of outstanding requests issued with **ChannelSequence** less than **Open.ChannelSequence**.
- **Open FileName**: A variable length string that contains the Unicode file name supplied by the client for opening the file.
- **Open DesiredAccess**: The access mode requested by the client while opening the file, in the format specified in section [2.2.13.1](#).

- **Open.ShareMode**: The sharing mode requested by the client while opening the file, in the format specified in section [2.2.13](#).
- **Open.CreateOptions**: The create options requested by the client while opening the file, in the format specified in section [2.2.13](#).
- **Open.FileAttributes**: The file attributes used by the client for opening the file, in the format specified in section [2.2.13](#).
- **Open.CreateDisposition**: The create disposition requested by the client for opening the file, in the format specified in section [2.2.13](#).

3.3.1.11 Per Lease Table

If the server implements the SMB 2.1 or SMB 3.0 dialect 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

If the server implements the SMB 2.1 or SMB 3.0 dialect 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.

If the server implements the SMB 3.0 dialect and supports leasing, it implements the following:

- **Lease.Epoch**: A sequence number incremented by the server on every lease state change.
- **Lease.Version**: A number indicating the lease version.

3.3.1.13 Per Request

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.[<156>](#)
- **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.

If the server implements the SMB 3.0 dialect, it MUST implement the following:

- **Request.IsEncrypted**: A Boolean that, if set, indicates that the request has been encrypted.

3.3.1.14 Per Channel

If the server implements the SMB 3.0 dialect, the server implements the following:

- **Channel.SigningKey**: The first 16 bytes of the cryptographic key for this authenticated channel. If the cryptographic key is less than 16 bytes, it is right-padded with zero bytes.
- **Channel.Connection**: The connection on which this channel was established.

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.<157>](#) If the server implements the SMB 2.1 or SMB 3.0 dialect, 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.[<158>](#) 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[<159>](#) 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[<160>](#) default value.
- **ServerSideCopyMaxChunkSize** MUST be set to an implementation-specific[<161>](#) default value.
- **ServerSideCopyMaxDataSize** MUST be set to an implementation-specific[<162>](#) default value.
- **ShareList** MUST be set to an empty list.

If the server implements the SMB 2.1 or SMB 3.0 dialect, it MUST initialize the following:

- **ServerHashLevel** MUST be set to an implementation-specific[<163>](#) default value.

If the server implements the SMB 2.1 or 3.0 dialect and supports leasing, the server MUST implement the following:

- **GlobalLeaseTableList** MUST be set to an empty list.
- **MaxResiliencyTimeout** SHOULD [<164>](#) be set to an implementation-specific default value.

If the server implements the SMB 3.0 dialect, the server MUST implement the following:

- **EncryptionAlgorithmList** MUST be initialized with an implementation-specific [<165>](#) list of encryption algorithms supported by the server.
- **EncryptData** MUST be set in an implementation-specific manner [<166>](#).
- **RejectUnencryptedAccess** MUST be set in an implementation-specific manner [<167>](#).

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 an 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**.
- For the command requests which include **FileId**, if **Connection.Dialect** is "3.000" and **ChannelSequence** is equal to **Open.ChannelSequence**, the server MUST decrement **Open.OutstandingRequestCount** by 1. Otherwise, the server MUST decrement **Open.OutstandingPreRequestCount** by 1.

3.3.4.1.1 Signing the Message

The server MUST sign the message under the following conditions:

- If the response message being sent contains a nonzero **SessionId** and a zero **TreeId** in the SMB2 header, and the session identified by **SessionId** has **Session.SigningRequired** equal to TRUE.
- If the response message being sent contains a nonzero **SessionId**, and a nonzero **TreeId** in the SMB2 header, and the session identified by **SessionId** has **Session.SigningRequired** equal to TRUE, and **Share.EncryptData** for the share associated with the **TreeId** has **TreeConnect.EncryptData** equal to FALSE.

- If the request was signed by the client, and the response is not an interim response to an asynchronously processed request.

If the response is an [SMB2_OPLOCK_BREAK](#) or an interim response to an asynchronously processed request, then the response SHOULD<168> be signed.

If **Connection.Dialect** is "3.000", and if the response being sent is an SMB2 SESSION_SETUP Response, the server MUST use **Session.SigningKey**. For all other responses the server MUST provide **Channel.SigningKey** by looking up the **Channel** in **Session.ChannelList**, where the connection matches the **Channel.Connection**.

Otherwise, the server MUST use **Session.SessionKey** for signing the response.

The server provides the key for signing, the length of the response, and the response itself, and calculates the signature as specified in section [3.1.4.1](#). If the server signs the request, it MUST set the SMB2_FLAGS_SIGNED bit in the **Flags** field of the SMB2 header.

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 the SMB 2.1 or SMB 3.0 dialect and the request is a multi-credit request, the server MUST consume multiple credits as specified in section [3.3.5.2.3](#). 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.<169>](#)

3.3.4.1.3 Sending Compounded Responses

The server MAY<170> 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** SHOULD<171> be set to 0. The server MAY<172> grant credits separately on each response 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.

3.3.4.1.4 Encrypting the Message

If **Connection.Dialect** is "3.000", the server MUST encrypt the message before sending, if any of the following conditions is satisfied:

- If the message being sent is any response to a client request for which **Request.IsEncrypted** is TRUE.
- If **Session.EncryptData** is TRUE and the response being sent is not SMB2_NEGOTIATE.
- If **Session.EncryptData** is FALSE, the response being sent is not SMB2_NEGOTIATE or SMB2_SESSION_SETUP or SMB2_TREE_CONNECT, and **Share.EncryptData** for the share associated with the **TreeId** in the SMB2 header of the response is TRUE.

If any of the above conditions are satisfied, the server MUST construct the SMB2 TRANSFORM_HEADER specified in section [2.2.41](#) as follows:

- **Nonce** is set to a newly generated implementation-specific value that is never reused for any encrypted message sent within the session.
- **OriginalMessageSize** is set to the size of the SMB2 message being sent.
- **SessionId** is set to **Session.SessionId**.
- **EncryptionAlgorithm** is set to a value specified in section [2.2.41](#), corresponding to the encryption algorithm chosen from **EncryptionAlgorithmList** using a local configuration policy. [<173>](#)
- **Signature** field is set to a value generated using the using the AES-CCM algorithm as specified in [\[RFC5084\]](#) with the following inputs:
 - The size of the SMB2 TRANSFORM_HEADER excluding the **ProtocolId** and **Signature** fields, as the optional authenticated data length.
 - The SMB2 TRANSFORM_HEADER excluding the **ProtocolId** and **Signature** fields, as the buffer to sign as the optional authenticated data.
 - The SMB2 message including the header and the payload, as the data to be signed.
 - **Session.EncryptionKey** as the key to be used for signing.

The server MUST encrypt the SMB2 message using the encryption algorithm chosen above and using **Session.EncryptionKey**.

The server MUST append the encrypted SMB2 message to the SMB2 TRANSFORM_HEADER and send it to the client.

3.3.4.2 Sending an Interim Response for an Asynchronous Operation

The server MAY[<174>](#) choose to send an interim response for any request that is received. It SHOULD[<175>](#) 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[<176>](#) 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~~<177>~~ fail requests in a compound chain that try to go async.
- The server MUST set the SMB2_FLAGS_SERVER_TO_REDIRECTION 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_REDIRECTION 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.<178>](#)
- 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.

If **Connection.Dialect** is "3.000", the server MUST return **Open.TreeConnect.Session.ApplicationKey**. Otherwise, the server MUST return **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 2.1.5.17.3.

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.Dialect** is "3.000" and **Open.Connection** are NULL, the server MUST select an alternate connection in **Session.ChannelList** and update **Open.Connection**. Otherwise, if 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, and the **MessageId** to 0xFFFFFFFFFFFFFF. 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.OlockState to Breaking and set **Open.OlockTimeout** to the current time plus an implementation-specific default value in milliseconds.[<179>](#) 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.Dialect** is "3.000" and **Open.Connection** is NULL, the server MUST select alternate connection in **Session.ChannelList** and update **Open.Connection**.

Otherwise, if **Lease.BreakToLeaseState** does not contain SMB2LEASEHANDLECACHING 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 SMB2OPLOCKBREAK, and the **MessageId** to 0xFFFFFFFFFFFFFF. The server MUST set the **SessionId** and **TreeId** in the [SMB2 header](#) to 0. If **Lease.LeaseState** is SMB2LEASEREADCACHING, the server MUST set the **Flags** field of the message to zero. Otherwise the server MUST set the **Flags** field of the message to SMB2NOTIFYBREAKLEASEFLAGACKREQUIRED, indicating to the client that lease acknowledgment is required. 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.OlockState** 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.[<180>](#) If **Connection.Dialect** is equal to "3.0", the server MUST set **NewEpoch** to **Lease.Epoch** + 1.

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. The server SHOULD [<181>](#) set STYPE_CLUSTER_FS, STYPE_CLUSTER_SOFS, and STYPE_CLUSTER_DFS in an implementation-defined manner.
- **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 the server implements the SMB 3.0 dialect, **Share.EncryptData** 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.

If the server implements the SMB 3.0 dialect, **Share.CATimeout** MUST be set to an implementation-specific value. [<182>](#)

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 the 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 the 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 the 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 the 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.
- **Share.IsCA** MUST be set to TRUE if SHI1005_flags contains SHI1005_FLAGS_ENABLE_CA bit as specified in [\[MS-SRVS\]](#) section 2.2.4.29; otherwise, it MUST be set to FALSE.
- **Share.EncryptData** MUST be set to TRUE if SHI1005_flags contains SHI1005_FLAGS_ENCRYPT_DATA bit as specified in [\[MS-SRVS\]](#) section 2.2.4.29. Otherwise, it MUST be set to FALSE.

3.3.4.15 Server Application DereRegisters 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
SHARE_INFO_503_I.shi503_servername	Share.ServerName
SHARE_INFO_503_I.shi503_security_descriptor	Share.FileSecurity
SHARE_INFO_1005.shi1005_flags	ShareFlags MUST be set based on the individual share properties: <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_ROOT bit if the per-share property Share.IsDfs is TRUE.

Output Parameters	SMB2 Share Properties
	<ul style="list-style-type: none"> ▪ The server MUST set the SHI1005_FLAGS_ACCESS_BASED_DIRECTORY_ENU M 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 **Open.Connection.Dialect** is not "2.002", the server 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. [<183>](#)
 - 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_numOpens	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 Session.IsGuest is TRUE
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
coni1_id	TreeConnect.TreeGlobalId
coni1_type	TreeConnect.Share.Type
coni1_numOpens	TreeConnect.OpenCount

Output Parameters	SMB2 TreeConnect Properties
coni1_num_users	1
coni1_time	Current time minus Treeconnect.CreationTime , in seconds.
coni1_username	TreeConnect.Session.UserName
coni1_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 <184> 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** <185> 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, **ServerEnabled** 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**.

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

STAT_SERVER_0 members	SMB2 ServerStatistics Properties
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 [<186>](#) as obtained by implementation-specific means from the transport that indicated the incoming connection.

Connection.SessionTable MUST be set to an empty table.

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**,TRUE>.

This connection MUST be inserted into the global **ConnectionList**.

3.3.5.2 Receiving Any Message

If the server implements the SMB 3.0 dialect, and the **ProtocolId** in the header of the received message is 0x424d53FD, the server MUST decrypt the message as specified in section [3.3.5.2.1](#) before performing the following steps.

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 [3.3.5.16.<187>](#)
- **Request.Open** MUST be set to NULL.
- If the server implements the SMB 3.0 dialect, **Request.IsEncrypted** MUST be initialized to FALSE. If the request was successfully received as encrypted as specified in section [3.3.5.2.1](#), **Request.IsEncrypted** MUST be set to TRUE.

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 Decrypting the Message

This section is applicable for only the SMB 3.0 dialect.

If the **ProtocolId** in the header of the received message is 0x424d53FD, the server MUST perform the following:

- If the size of the message received from the client is not greater than the size of SMB2 TRANSFORM_HEADER as specified in section [2.2.41](#), the server MUST fail the request with STATUS_BUFFER_OVERFLOW.
- If **OriginalMessageSize** value received in the TRANSFORM_HEADER is greater than the implementation-specific limit [<188>](#) or if it is less than the size of SMB2 Header, the server MUST fail the request with STATUS_BUFFER_OVERFLOW.
- If the **EncryptionAlgorithm** in the SMB2 TRANSFORM_HEADER is not a valid algorithm listed in **EncryptionAlgorithmList**, the server MUST fail the request with STATUS_INVALID_PARAMETER.

- The server MUST look up the session in the **Connection.SessionTable** using the **SessionId** in the SMB2 TRANSFORM_HEADER of the request. If the session is not found, the server MUST fail the request with STATUS_USER_SESSION_DELETED.
- The server MUST decrypt the message using **Session.DecryptionKey** and the algorithm specified by the **EncryptionAlgorithm** in TRANSFORM_HEADER, and pass in the TRANSFORM_HEADER, excluding the **Signature** and **ProtocolId** fields, as the Optional Authenticated Data input for AES-CCM. If decryption succeeds, the server compares the signature in the transform header with the signature returned by the decryption algorithm. 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 the signature verification succeeds, the server MUST continue processing the decrypted packet, as specified in subsequent sections.

3.3.5.2.2 Verifying the Connection State

If the request being received is not an [SMB2 NEGOTIATE Request](#) or a traditional SMB_COM_NEGOTIATE, as described in section [1.7](#), and **Connection.NegotiateDialect** is 0xFFFF or 0x02FF, the server MUST disconnect the connection, as specified in section [3.3.7.1](#), and send no reply.

3.3.5.2.3 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,<189>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**.

3.3.5.2.4 Verifying the Signature

If **Connection.Dialect** is "3.000" and if the decryption in section [3.3.5.2.1](#) succeeds, the server MUST skip the processing in this section.

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](#).

If **Connection.Dialect** is "3.000", the server MUST use **Session.SigningKey** if the request is for binding a session, and for all other requests the server MUST use **Channel.SigningKey** in

Session.ChannelList, where **Channel.Connection** matches the connection on which the request is received.

Otherwise, the server MUST use **Session.SessionKey** as the session key to verify the signature.

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.[<190>](#)

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.SigningRequired** is equal to TRUE, the server MUST fail this request with STATUS_ACCESS_DENIED. The server MAY[<191>](#) also disconnect the connection, as specified in section 3.3.7.1. If either the session is not found, or **Session.SigningRequired** 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.

3.3.5.2.5 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[<192>](#) fail the request with the error code STATUS_INVALID_PARAMETER.

3.3.5.2.6 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[<193>](#) 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.

3.3.5.2.7 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[<194>](#) 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[<195>](#) fail each of the requests with STATUS_INVALID_PARAMETER.

3.3.5.2.7.1 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. The server MAY send responses to unrelated compounded requests separately.

3.3.5.2.7.2 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[<196>](#) 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[<197>](#) 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.

3.3.5.2.8 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.

3.3.5.2.9 Verifying the Session

The server MUST look up the **Session** in **Connection.SessionTable** by using the **SessionId** in the SMB2 header of the request. If no session is found, the server MUST fail the request with STATUS_USER_SESSION_DELETED.

If a session is found and **Session.State** is set to Expired, the server SHOULD fail the request with STATUS_NETWORK_SESSION_EXPIRED. If **Session.State** is InProgress, the server MUST fail the request with STATUS_ACCESS_DENIED.

If **Connection.Dialect** is "3.000", **Session.EncryptData** is TRUE, and **RejectUnencryptedAccess** is TRUE, the server MUST locate the **Request** in **Connection.RequestList** for which **Request.MessageId** matches the **MessageId** value in the SMB2 header of the request. If **Request.IsEncrypted** is FALSE, the server MUST fail the request with STATUS_ACCESS_DENIED.

3.3.5.2.10 Verifying the Channel Sequence Number

This section is applicable only for the SMB 3.0 dialect and for the command requests which include **FileId**.

If the SMB2_FLAGS_REPLAY_OPERATION bit is not set in the **Flags** field of the [SMB2 Header](#):

- If **ChannelSequence** in the SMB2 Header is equal to **Open.ChannelSequence**, the server MUST increment **Open.OutstandingRequestCount** by 1.
- Otherwise, if the unsigned difference using 16-bit arithmetic between **ChannelSequence** and **Open.ChannelSequence** is less than or equal to 0x7FFF, the server MUST increment **Open.OutstandingPreRequestCount** by **Open.OutstandingRequestCount**, and MUST set **Open.OutstandingRequestCount** to 1. The server MUST set **Open.ChannelSequence** to **ChannelSequence** in the SMB2 Header.
- Otherwise, the server MUST fail SMB2 WRITE, SET_INFO, and IOCTL requests with STATUS_FILE_NOT_AVAILABLE.

If the SMB2_FLAGS_REPLAY_OPERATION bit is set in the Flags field of the SMB2 Header:

- If **ChannelSequence** in the SMB2 Header is equal to **Open.ChannelSequence** and the following:
 - If **ChannelSequence** in the SMB2 Header is equal to **Open.ChannelSequence** and **Open.OutstandingPreRequestCount** is equal to zero, the server MUST increment **Open.OutstandingRequestCount** by 1.
 - Otherwise, if the unsigned difference using 16-bit arithmetic between **ChannelSequence** and **Open.ChannelSequence** is less than or equal to 0x7FFF and **Open.OutstandingPreRequestCount** is equal to zero, the server MUST increment **Open.OutstandingPreRequestCount** by **Open.OutstandingRequestCount** and MUST set **Open.OutstandingRequestCount** to 1. The server MUST set **Open.ChannelSequence** to **ChannelSequence** in the SMB2 Header.
- Otherwise, the server MUST fail SMB2 WRITE, SET_INFO, and IOCTL requests with STATUS_FILE_NOT_AVAILABLE.

3.3.5.2.11 Verifying the Tree Connect

The server MUST look up the **TreeConnect** in **Session.TreeConnectTable** by using the **TreeId** in the SMB2 header of the request. If no tree connect is found, the request MUST be failed with STATUS_NETWORK_NAME_DELETED.

If **Connection.Dialect** is "3.000", **TreeConnect.Share.EncryptData** is TRUE, and **RejectUnencryptedAccess** is TRUE, the server MUST locate the **Request** in **Connection.RequestList** for which **Request.MessageId** matches the **MessageId** value in the

SMB2 header of the request. If **Request.IsEncrypted** is FALSE, the server MUST fail the request with STATUS_ACCESS_DENIED.

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 or SMB 3.0 Support

If the server does not implement the SMB 2.1 or 3.0 dialect, processing MUST continue as specified in [3.3.5.3.2](#).

If the server implements the SMB 2.1 or 3.0 dialect, 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 the underlying connection is either TCP port 445 or RDMA, **Connection.SupportsMultiCredit** MUST be set to TRUE.

The server MUST set the command of the SMB2 header to SMB2 NEGOTIATE. All other values MUST be set following the syntax specified in section [2.2.1](#), and any value not defined there with a default MUST be set to 0. The header is followed by an SMB2 NEGOTIATE Response that MUST be constructed as specified in [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 [`<199>`](#), 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. [`<200>`](#)

- **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.[<201>](#)
- **SystemTime** is set to the current time, in **FILETIME** format as specified in [[MS-DTYP](#)] section [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".[<202>](#) 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-SMB1](#)], 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. 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.[<203>](#)
- **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[<204>](#), 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, 0x0210, or 0x0300, the server MUST disconnect the connection, as specified in section [3.3.7.1](#), and not reply.

The server MUST set **Connection.ClientCapabilities** to the capabilities received in the SMB2 NEGOTIATE Request.

If the server implements the SMB2.1 or 3.0 dialect, 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.

If the **DialectCount** of the SMB2 NEGOTIATE request is 0, the server MUST fail the request with STATUS_INVALID_PARAMETER.

The server MUST select the greatest common dialect between the dialects it implements 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", "2.100", or "3.000", and **Connection.NegotiateDialect** to 0x0202, 0x0210, or 0x0300 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 or 3.0 and the underlying connection is either TCP port 445 or RDMA, **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** field.
- **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.
 - SMB2_GLOBAL_CAP_MULTI_CHANNEL if **Connection.Dialect** is "3.000", the server supports multichannel, and SMB2_GLOBAL_CAP_MULTI_CHANNEL is set in the **Capabilities** field of the request.
 - SMB2_GLOBAL_CAP_DIRECTORY_LEASING if **Connection.Dialect** is "3.000", the server supports directory leasing, and SMB2_GLOBAL_CAP_DIRECTORY_LEASING is set in the **Capabilities** field of the request.
 - SMB2_GLOBAL_CAP_PERSISTENT_HANDLES if **Connection.Dialect** is "3.000", SMB2_GLOBAL_CAP_PERSISTENT_HANDLES is set in the **Capabilities** field of the request, and the server supports persistent handles.
 - SMB2_GLOBAL_CAP_ENCRYPTION if **Connection.Dialect** is "3.000", **EncryptData** is TRUE, and SMB2_GLOBAL_CAP_ENCRYPTION is set in the **Capabilities** field of the request.
- **MaxTransactSize** is set to the maximum buffer size [<205>](#), 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

If the server implements the SMB 3.0 dialect, the server MUST store the value of the **SecurityMode** field in **Connection.ServerSecurityMode** and MUST store the value of the **Capabilities** field in **Connection.ServerCapabilities**.

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 the server implements the SMB 3.0 dialect, **Connection.Dialect** is not "3.000", **EncryptData** is TRUE, and **RejectUnencryptedAccess** is TRUE, the server MUST fail the request with STATUS_ACCESS_DENIED.
2. If **Connection.Dialect** is 3.000, **EncryptData** is TRUE, **RejectUnencryptedAccess** is TRUE, and **Connection.ClientCapabilities** does not include the SMB2_GLOBAL_CAP_ENCRYPTION bit, the server MUST fail the request with STATUS_ACCESS_DENIED.
3. 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](#).
4. The server MUST look up the session in the **GlobalSessionTable** using the **SessionId** from the SMB2 header.
5. If the session is not found, the server MUST fail the session setup request with STATUS_USER_SESSION_DELETED. Otherwise, proceed to step 6.
6. 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 7.
7. If **Connection.Dialect** is "3.000", and the SMB2_SESSION_FLAG_BINDING is set in the **Flags** field of the request, the server MUST do the following. Otherwise, proceed to step 8.

- If **Connection.Dialect** is not same as **Session.Connection.Dialect**, the server MUST fail the request with STATUS_INVALID_PARAMETER.
- If the SMB2_FLAGS_SIGNED bit is not set in the **Flags** field in the header, the server MUST fail the request with error STATUS_INVALID_PARAMETER.
- If **Session.State** is InProgress, the server MUST fail the request with STATUS_REQUEST_NOT_ACCEPTED.
- If **Session.State** is Expired, the server MUST fail the request with STATUS_NETWORK_SESSION_EXPIRED.
- If **Session.IsAnonymous** or **Session.IsGuest** is TRUE, the server MUST fail the request with STATUS_NOT_SUPPORTED.
- If there is a session in **Connection.SessionTable** identified by the **SessionId** in the request, the server MUST fail the request with STATUS_REQUEST_NOT_ACCEPTED.
- The server MUST verify the signature as specified in section [3.3.5.2.4](#), using the **Session.SessionKey**.
- The server MUST obtain the security context from the GSS authentication subsystem, and it MUST invoke the GSS_Inquire_context call as specified in [\[RFC2743\]](#) section 2.2.6, passing the security context as the input parameter. If the returned "src_name" does not match with the **Session.Username**, the server MUST fail the request with error code STATUS_NOT_SUPPORTED.

8. If **Session.State** is Valid, the server MUST do the following:

- If **Connection.Dialect** is "2.002", the server MUST fail the session setup request with STATUS_REQUEST_NOT_ACCEPTED.
- Otherwise, the server MUST process the session setup request as specified in section [3.3.5.2](#).

9. The server MUST continue processing the request as specified in section [3.3.5.3](#).

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 session MUST be inserted into **Connection.SessionTable**. 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.SigningRequired** 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.
- If **Connection.Dialect** is "3.000", **Session.EncryptData** is set to global **EncryptData**.

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.[<206>](#)

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_pwerrors** 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_REDİR 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 **SecurityBufferSizeOffset** 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 the server MAY 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, [<207>](#) the server MUST set the SMB2_SESSION_FLAG_IS_GUEST in the **SessionFlags** field of the SMB2 SESSION_SETUP Response and MUST set **Session.IsGuest** to TRUE.

5. **Session.SigningRequired** MUST be set to TRUE under the following conditions:

- If the SMB2_NEGOTIATE_SIGNING_REQUIRED bit is set in the **SecurityMode** field of the client request.
- If the SMB2_SESSION_FLAG_IS_GUEST bit is not set in the **SessionFlags** field and **Session.IsAnonymous** is FALSE and either **Connection.ShouldSign** or global **RequireMessageSigning** is TRUE.

6. The server MUST query the session key for this authentication from the underlying authentication protocol and store the session key in **Session.SessionKey**, if **Session.SessionKey** is NULL. **Session.SessionKey** MUST be set as specified 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. When NTLM authentication via GSS-API is used, **Session.SessionKey** MUST be set to **ExportedSessionKey**, see [\[MS-NLMP\]](#) section 3.1.5.1.

7. If **Connection.Dialect** is "3.000", the server MUST generate **Session.SigningKey** as specified in section [3.1.4.2](#) by providing the following inputs:

- **Session.SessionKey** as the key derivation key.
- The case-sensitive ASCII string "SMB2AESEMAC" as the label.
- The label buffer size in bytes, including the terminating null character. The size of "SMB2AESEMAC" is 12.
- The case-sensitive ASCII string "SmbSign" as context for the algorithm.
- The context buffer size in bytes, including the terminating null character. The size of "SmbSign" is 8.

8. If **Connection.Dialect** is "3.000", **Session.ApplicationKey** MUST be generated as specified in section [3.1.4.2](#) and passing the following inputs:

- **Session.SessionKey** as the key derivation key.
- The case-sensitive ASCII string "SMB2APP" as the label.
- The label buffer size in bytes, including the terminating null character. The size of "SMB2APP" is 8.
- The case-sensitive ASCII string "SmbRpc" as context for the algorithm.
- The context buffer size in bytes, including the terminating null character. The size of "SmbRpc" is 7.

9. If **Connection.Dialect** is "3.000" and if the SMB2_SESSION_FLAG_BINDING is set in the **Flags** field of the request, the server MUST insert the Session into **Connection.SessionTable**, and insert a new **Channel** entry in **Session.ChannelList** with the following values:
- **Channel.SigningKey** MUST be set to the **Session.SigningKey**.
 - **Channel.Connection** MUST be set to the connection on which this request is received.
10. If **Connection.Dialect** is "3.000", global **EncryptData** is TRUE, and **Connection.ClientCapabilities** includes the SMB2_GLOBAL_CAP_ENCRYPTION bit, the server MUST do the following:
- Set the SMB2_SESSION_FLAG_ENCRYPT_DATA flag in the **SessionFlags** field of the SMB2 SESSION_SETUP Response.
 - Set **Session.SigningRequired** to FALSE.
 - Generate **Session.EncryptionKey** and **Session.DecryptionKey** as specified in section [3.1.4.2](#) by providing the following inputs:
 - **Session.SessionKey** as the key derivation key.
 - The case-sensitive ASCII string "SMB2AESCCM" as the label.
 - The label buffer length in bytes, including the terminating null character. The size of "SMB2AESCCM" is 11.
 - The case-sensitive ASCII string as key derivation context. For generating the encryption key, this MUST be "ServerOut". For generating the decryption key, this MUST be "ServerIn"; note the blank space at the end.
 - The context buffer size in bytes, including the terminating null character. For generating both the encryption key and decryption key, the string size is 10.
11. If **Session.SigningRequired** is TRUE, the server MUST sign the final session setup response before sending it to the client. Otherwise, if **Connection.Dialect** is "3.000", and if the SMB2_SESSION_FLAG_BINDING is set in the **Flags** field of the request, the server MUST sign the response using **Channel.SigningKey**.
12. If the **PreviousSessionId** field of the request is not equal to zero and **PreviousSessionId** is not equal to the **SessionId** in the SMB2 header of the request, the server MUST take the following actions:
1. The server MUST look up the old session in **GlobalSessionTable**, where **Session.SessionId** matches **PreviousSessionId**. 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 also from the **Connection.SessionTable**, as specified in section [3.3.7.1](#).
 2. If the server determines that the authentications were for different users, the server MUST ignore the **PreviousSessionId** value.

13. **Session.State** MUST be set to Valid.

14. **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.
- If **Connection.Dialect** is "3.000", and if the SMB2_SESSION_FLAG_BINDING is set in the **Flags** field of the request, the server MUST sign the response by using **Session.SessionKey**.

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, as specified in section [3.3.5.2.9](#).

The server MUST remove this session from the **GlobalSessionTable** and also from the **Connection.SessionTable**, 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_opens MUST be decreased by 1. The server MUST close every **Open** in **Session.OpenTable** of the old session, where **Open.IsDurable** is FALSE, as specified in section [3.3.4.17](#). For all opens in **Session.OpenTable** where **Open.IsDurable** is TRUE, the server MUST set **Open.Session**, **Open.Connection** and **Open.TreeConnect** to NULL. 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. If **Connection.Dialect** is "3.000", each channel in **Session.ChannelList** MUST be removed and freed. 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, as specified in section [3.3.5.2.9](#).

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 do the following:

If the server implements the SMB 3.0 dialect, **Connection.Dialect** is not "3.000", **Share.EncryptData** is TRUE, and **RejectUnencryptedAccess** is TRUE, the server MUST fail the request with STATUS_ACCESS_DENIED.

If **Connection.Dialect** is "3.000", **Share.EncryptData** is TRUE, **RejectUnencryptedAccess** is TRUE, and **Connection.ClientCapabilities** does not include SMB2_GLOBAL_CAP_ENCRYPTION, the server MUST fail the request with STATUS_ACCESS_DENIED.

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 total number of current uses is equal to or greater than **Share.MaxUses**, the server MUST fail the request with STATUS_REQUEST_NOT_ACCEPTED.

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**, **Share.EncryptData**.)
 - The server MUST set all flags contained in **Share.CscFlags**.
 - The server SHOULD [<208>](#) set the SMB2_SHAREFLAG_DFS bit if the per-share property **Share.IsDfs** is TRUE, indicating that the share is part of a DFS namespace.

- The server SHOULD [<209>](#) set the SMB2_SHAREFLAG_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 SMB2_SHAREFLAG_ACCESS_BASED_DIRECTORY_ENUM bit if **Share.DoAccessBasedDirectoryEnumeration** is TRUE and **ServerHashLevel** is not **HashDisableAll**.
- The server MUST set the SMB2_SHAREFLAG_ALLOW_NAMESPACE_CACHING bit if **Share.AllowNamespaceCaching** is TRUE.
- The server MUST set the SMB2_SHAREFLAG_FORCE_SHARED_DELETE bit if **Share.ForceSharedDelete** is TRUE.
- The server MUST set the SMB2_SHAREFLAG_RESTRICT_EXCLUSIVE_OPENS bit if **Share.RestrictExclusiveOpens** is TRUE.
- If **Connection.Dialect** is "3.000", and **Share.EncryptData** is TRUE, the server MUST do the following:
 - Set the SMB2_SHAREFLAG_ENCRYPT_DATA bit.
 - If **Session.EncryptData** is FALSE, the server MUST generate an encryption key, and a decryption key as specified in section [3.1.4.2](#) by providing the following inputs and store them in **Session.EncryptionKey** and **Session.DecryptionKey**.
 - **Session.SessionKey** as the key derivation key.
 - The case-sensitive ASCII string "SMB2AESCCM" as the label
 - The label buffer length in bytes, including the terminating null character. The size of "SMB2AESCCM" is 11.
 - The case-sensitive ASCII string as key derivation context. For generating the encryption key, this MUST be "ServerOut". For generating the decryption key, this MUST be "ServerIn "; note the blank space at the end.
 - The context buffer size in bytes, including the terminating null character. For generating both the encryption key and decryption key, the string size is 10.
- If **Share.HashEnabled** is TRUE and **ServerHashLevel** is not **HashDisableAll**.
 - If **Connection.Dialect** is "3.000", the server MUST set the SMB2_SHAREFLAG_ENABLE_HASH_V1 and SMB2_SHAREFLAG_ENABLE_HASH_V2 bits in an implementation-specific manner. [<210>](#)
 - Otherwise, it SHOULD [<211>](#) set the SMB2_SHAREFLAG_ENABLE_HASH_V1 bit.
- The server MUST set the SMB2_SHAREFLAG_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, the server MUST set the SMB2_SHARE_CAP_DFS bit in the **Capabilities** field.
- If **Connection.Dialect** is "3.000" and **Share.IsCA** is TRUE, the server MUST set the SMB2_SHARE_CAP_CONTINUOUS_AVAILABILITY bit in the **Capabilities** field.
- If **TreeConnect.Share.Type** includes STYPE_CLUSTER_SOFS, the server MUST set the SMB2_SHARE_CAP_SCALEOUT bit in the **Capabilities** field.
- If **TreeConnect.Share.Type** includes STYPE_CLUSTER_FS, STYPE_CLUSTER_SOFS, or STYPE_CLUSTER_DFS, the server MUST set the SMB2_SHARE_CAP_CLUSTER bit in the **Capabilities** field.
- **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, as specified in section [3.3.5.2.9](#).

Tree Connect Verification:

The server MUST locate the tree connection, as specified in section [3.3.5.2.11](#).

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.

If **Connection.Dialect** is "3.000" and the request does not contain SMB2_CREATE_DURABLE_HANDLE_RECONNECT Create Context or SMB2_CREATE_DURABLE_HANDLE_RECONNECT_V2 Create Context, the server MUST look up an existing open in the **GlobalOpenTable** where **Open.FileName** matches the file name in the **Buffer** field of the request. If an **Open** entry is found, if **Open.IsPersistent** is TRUE and if **Open.Connection** is NULL, the server MUST fail the request with STATUS_FILE_NOT_AVAILABLE.

Session Verification:

The server MUST locate the session, as specified in section [3.3.5.2.9](#).

Tree Connect Verification:

The server MUST locate the tree connection, as specified in section [3.3.5.2.11](#).

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 path separator character, the server MUST fail the request with STATUS_INVALID_PARAMETER. 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.

For pipe opens, the server MUST ignore **FileAttributes**.

For print files, if the **FileAttributes** field includes FILE_ATTRIBUTE_DIRECTORY, the server MUST fail the open with the error code STATUS_NOT_SUPPORTED.

If any of the bits in the mask 0x0CE0FE00 are set in the **DesiredAccess** field, the server SHOULD^{<212>} 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 **ServerStatistics.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^{<213>} 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^{<214>} 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^{<215>} fail any request having a create context not specified in section [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 [<216>](#) 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.ForceLevel2Olock** is TRUE, and **RequestedOlockLevel** is SMB2_OPLOCK_LEVEL_BATCH or SMB2_OPLOCK_LEVEL_EXCLUSIVE, the server SHOULD [<217>](#) set **RequestedOlockLevel** to SMB2_OPLOCK_LEVEL_II.
- If **Connection.Dialect** is "3.000" **TreeConnect.Share.Type** includes STYPE_CLUSTER_SOFS and the **RequestedOlockLevel** is SMB2_OPLOCK_LEVEL_BATCH, the server MUST set **RequestedOlockLevel** 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 [\[MS-FSA\]](#). [<218>](#) 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](#). [<219>](#)

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.ClientGuid** is set to **Open.Connection.ClientGuid**.
- **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.OpslockLevel** is set to SMB2_OPLOCK_LEVEL_NONE.
- **Open.OpslockState** is set to None.
- **Open.OpslockTimeout** 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 **Connection.Dialect** is not "2.002" and the server 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.

If **Connection.Dialect** is "3.000", the server MUST initialize the following:

- **Open.CreateGuid** MUST be set to NULL.
- **Open.AppInstanceId** MUST be set to NULL.
- **Open.IsPersistent** MUST be set to FALSE.
- **Open.FileName** MUST be set to the file name in the **Buffer** field of the request.
- **Open.DesiredAccess** MUST be set to the **DesiredAccess** field of the request.
- **Open.ShareMode** MUST be set to the **ShareAccess** field of the request.
- **Open.CreateOptions** MUST be set to the **CreateOptions** field of the request.
- **Open.FileAttributesOptions** MUST be set to the **FileAttributes** field of the request.
- **Open.CreateDisposition** MUST be set to the **CreateDisposition** field of the request.

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 **Connection.Dialect** is "2.002" or the server does not support leasing, and **RequestedOplockLevel** is set to SMB2_OPLOCK_LEVELLEASE, the request MUST be failed with STATUS_NOT_SUPPORTED.

If **Connection.Dialect** is "2.100" or "3.000", the server supports leasing, and the **RequestedOplockLevel** is set to SMB2_OPLOCK_LEVELLEASE, 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.[220](#) 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.[221](#)
- **LastAccessTime** is set to the value queried from the object store for when the object was last accessed.[222](#)

- **LastWriteTime** is set to the value queried from the object store for when the object was last written to.[<223>](#)
- **ChangeTime** is set to the value queried from the object store for when the object was last modified, including attribute changes.[<224>](#)
- **AllocationSize** is set to the amount of space reserved for the object, in bytes, on the underlying object store.[<225>](#) If this is a named pipe, **AllocationSize** SHOULD be 0.[<226>](#)
- **EndOfFile** is set to the size of the **main stream** of the object in bytes.[<227>](#) For named pipes this value SHOULD be 0.[<228>](#)
- **FileAttributes** MUST be set to the attributes of the object following the syntax specified in section [2.2.14](#).[<229>](#)
- **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** (section [3.3.5.9.7](#)) is received, the server SHOULD[<230>](#) ignore any other create contexts except **SMB2_CREATE_REQUEST_LEASE** and **SMB2_CREATE_REQUESTLEASE_V2** 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 can be combined with any of those listed here except [SMB2_CREATE_DURABLE_HANDLE_RECONNECT](#).

The processing changes involved for this create context are:

In the "Open Execution" phase, the server MUST pass the received extended attributes array to the underlying object store to be stored on the created file.[<231>](#) 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.[<232>](#) If the object store does not support file security, the value MAY[<233>](#) 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.[<234>](#) 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.[<235>](#) 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.[<236>](#) 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.[<237>](#)

If **Connection.Dialect** is "3.000", the server MUST set the SMB2_CREATE_FLAG_REPARSEPOINT bit in **Flags** field in SMB2 CREATE response.

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, the server 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 fields **CreateContextLength** and **CreateContextOffset**. This structure MUST have the following values set:

- If the **ChangeTime** is not equal to the Timestamp 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. [<238>](#)
- If the **ChangeTime** is equal to the Timestamp in the request create context, the server MUST set **QueryStatus** to STATUS_NONE_MAPPED and **MaximalAccess** to zero.

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.

If the create request also includes an **SMB2_CREATE_DURABLE_HANDLE_RECONNECT** create context, the server MUST process the create context as specified in section [3.3.5.9.7](#) and skip this section.

If the create request also includes an **SMB2_CREATE_DURABLE_HANDLE_REQUEST_V2** or **SMB2_CREATE_DURABLE_HANDLE_RECONNECT_V2** create context, the server SHOULD [<239>](#) fail the create request with STATUS_INVALID_PARAMETER.

If an **SMB2_CREATE_REQUESTLEASE** Create Context or an **SMB2_CREATE_REQUESTLEASE_V2** Create Context is also present in the request and the lease is being requested on a directory, the server MUST ignore this **SMB2_CREATE_DURABLE_HANDLE_REQUEST** Create Context and skip this section.

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 the create request also includes an SMB2_CREATE_DURABLE_HANDLE_REQUEST create context, the server MUST ignore the SMB2_CREATE_DURABLE_HANDLE_REQUEST create context.

If the create request also contains an SMB2_CREATE_DURABLE_HANDLE_REQUEST_V2 or SMB2_CREATE_DURABLE_HANDLE_RECONNECT_V2 create context, the server SHOULD [<240>](#) fail the request with STATUS_INVALID_PARAMETER.

If an SMB2_CREATE_REQUESTLEASE_V2 create context is also present in the request, **Connection.Dialect** is "3.000", the server supports directory leasing, **Open.Lease** is not NULL, and **Lease.LeaseKey** does not match the **LeaseKey** provided in the SMB2_CREATE_REQUESTLEASE_V2 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](#).

If an [SMB2_CREATE_REQUESTLEASE create context](#) is also present in the request, **Connection.Dialect** is "2.100" or "3.000", the server supports leasing, **Open.Lease** is not NULL, and **Lease.LeaseKey** does not match the **LeaseKey** provided in the SMB2_CREATE_REQUESTLEASE 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.ClientGuid** is not equal to the **ClientGuid** of the connection that received this request, the server MUST fail the create request with STATUS_OBJECT_NAME_NOT_FOUND.
4. 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](#). If **Open.Session** is not NULL, the server MUST fail the request with STATUS_OBJECT_NAME_NOT_FOUND.
5. If the user represented by **Session.SecurityContext** is not the same user denoted by **Open.DurableOwner**, the server MUST fail the request with STATUS_ACCESS_DENIED and proceed as specified in "Failed Open Handling" in section [3.3.5.9](#).
6. The server MUST set the **Open.Connection** to refer to the connection that received this request.
7. The server MUST set the **Open.Session** to refer to the session that received this request.
8. 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.
9. The server MUST regenerate **Open.FileId** (the volatile portion of the [SMB2_FILEID](#)).

10.The server MUST insert the open into the **Session.OpenTable** with the **Open.FileId** as the new key.

11.The "Successful Open Initialization" and "Oplock Acquisition" phases MUST be skipped, and processing MUST continue as specified in "Response Construction".

12.In the "Response Construction" phase:

If **Connection.Dialect** is "3.000" and the server supports directory leasing, **Open.Lease** is not NULL, and **Lease.Version** is 2, then the server MUST construct an SMB2_CREATE_RESPONSELEASE_V2 create context, following the syntax specified in section [2.2.14.2.11](#), and include it in the buffer described by the response **CreateContextLength** and **CreateContextOffset** fields. This structure MUST have the following values set:

- **LeaseKey** MUST be set to **Lease.LeaseKey**.
- **LeaseState** MUST be set to **Lease.LeaseState**.
- **ParentLeaseKey** MUST be set to the **ParentLeaseKey** in the request.

If **Connection.Dialect** is "3.000" and the server supports leasing, **Open.Lease** is not NULL, and **Lease.Version** is 1, then the server MUST construct an [SMB2_CREATE_RESPONSELEASE](#) Create Context, following the syntax specified in section [2.2.14.2.10](#), and include it in the buffer described by the response **CreateContextLength** and **CreateContextOffset** fields. This structure MUST have the following values set:

- **LeaseKey** MUST be set to **Lease.LeaseKey**.
- **LeaseState** MUST be set to **Lease.LeaseState**.

If **Connection.Dialect** is not "2.100", the server supports leasing, and **Open.Lease** is not NULL, then the server MUST construct an SMB2_CREATE_RESPONSELEASE create context, following the syntax specified in section [2.2.14.2.10](#), 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_REQUESTLEASE Create Context

This section applies only to servers that implement the SMB 2.1 or 3.0 dialect. If the name of the create context, as defined in section [2.2.13.2](#), is "Rqls" and the **DataLength** field equals 0x20, the server MUST process the create context as SMB2_CREATE_REQUESTLEASE.

If both SMB2_CREATE_DURABLE_HANDLE_RECONNECT and SMB2_CREATE_REQUESTLEASE 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 support leasing, or if the **RequestedOplockLevel** is not **SMB2_OPLOCK_LEVELLEASE**, the server MUST ignore the [SMB2_CREATE_REQUESTLEASE Create Context](#) request.

By specifying a **RequestedOplockLevel** of **SMB2_OPLOCK_LEVELLEASE**, the client is requesting that a lease be acquired for this open. If the request does not provide an

SMB2_CREATE_REQUESTLEASE Create Context, the lease request MUST be ignored and **Open.OlockLevel** 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_REQUESTLEASE](#) 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_REQUESTLEASE** 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 **Connection.Dialect** is "3.000", **Lease.Version** is set to 1.

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 Olock Acquisition phase.

The caching state requested in **LeaseState** of the **SMB2_CREATE_REQUESTLEASE** SHOULD contain a valid **LeaseState** as specified in [3.3.1.12](#). The server MUST ignore the undefined bits in **LeaseState**.

During "Olock 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 "Olock Acquisition". If the underlying object store does support leasing, the following steps are taken:

If **TreeConnect.Share.ForceLevel2Olock** is TRUE, and **LeaseState** includes **SMB2LEASE_WRITE_CACHING**, the server MUST clear the bit **SMB2LEASE_WRITE_CACHING** in the **LeaseState** field.

If **Connection.Dialect** is "3.000" **TreeConnect.Share.Type** includes STYPE_CLUSTER_SOFS, and if **LeaseState** includes SMB2_LEASE_READ_CACHING, the server MUST set **LeaseState** to SMB2_LEASE_READ_CACHING, otherwise set **LeaseState** to SMB2_LEASE_NONE.

If the caching state requested in **LeaseState** of the **SMB2_CREATE_REQUESTLEASE** 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.[<241>](#) 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_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_RESPONSELEASE** response create context, following the syntax specified in section [2.2.14.2.10](#), 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.OlockState** to Held, set **Open.Lease** to a reference to Lease, set **Open.OlockLevel** to SMB2_OPLOCK_LEVELLEASE, 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.[<242>](#) 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 [2.2.14.2.9](#), and include it in the buffer described by the response's **CreateContextLength** and **CreateContextOffset**.

3.3.5.9.10 Handling the **SMB2_CREATE_DURABLE_HANDLE_REQUEST_V2** Create Context

This section applies only to servers that implement the SMB 3.0 dialect.

If the create request also includes an **SMB2_CREATE_DURABLE_HANDLE_REQUEST** create context, or an **SMB2_CREATE_DURABLE_HANDLE_RECONNECT** or **SMB2_CREATE_DURABLE_HANDLE_RECONNECT_V2** create context, the server MUST fail the create request with STATUS_INVALID_PARAMETER.

If the **SMB2_FLAGS_REPLY_OPERATION** bit is set in the SMB2 header, the client is requesting that the **Open** be marked for replay operation. The server MUST locate the **Open** in **GlobalOpenTable** where **Open.CreateGuid** matches the **CreateGuid** in **SMB2_DURABLE_HANDLE_REQUEST_V2** create context.

If an **Open** is not found, or if the SMB2_FLAGS_REPLAY_OPERATION bit is not set in the SMB2 header, the server MUST continue the create process specified in the "Open Execution" Phase, and perform the following additional steps:

- The server MUST set **Open.CreateGuid** to the **CreateGuid** in SMB2_CREATE_DURABLE_HANDLE_REQUEST_V2.
- In the "Successful Open Initialization" phase, the server MUST set **Open.IsDurable** to TRUE. The server MUST also set **Open.DurableOwner** to a security descriptor accessible only by the user represented by **Open.Session.SecurityContext**. If the SMB2_DHANDLE_FLAG_PERSISTENT bit is set in the Flags field of the request and if **TreeConnect.Share.IsCA** is TRUE, the server MUST set **Open.IsPersistent** to TRUE.

If an **Open** is found, the server MUST verify the following:

- If **Open.IsDurable** is FALSE or **Open.Lease** is not equal to the **LeaseState** specified in the SMB2_CREATE_REQUESTLEASE or SMB2_CREATE_REQUESTLEASE_V2 Create Context or **Open.DurableOwner** is not the user represented by **Session.SecurityContext**, the server MUST fail the request with STATUS_ACCESS_DENIED.
- If **Open.FileAttributes** does not match the **FileAttributes** field of the SMB2 CREATE request, the server MUST fail the request with STATUS_INVALID_PARAMETER.
- If **Open.CreateDisposition** does not match the **CreateDisposition** field of the SMB2 CREATE request, the server MUST fail the request with STATUS_INVALID_PARAMETER.
- If **Open.IsPersistent** is TRUE and the SMB2_DHANDLE_FLAG_PERSISTENT bit is not set in the **Flags** field of the SMB2_CREATE_DURABLE_HANDLE_REQUEST_V2 Create Context, the server MUST fail the request with STATUS_INVALID_PARAMETER.

The server MUST construct the create response from **Open**, as specified in the "Response Construction" phase, with the following additional steps, and send the response to client.

The server MUST skip the construction of the SMB2_CREATE_DURABLE_HANDLE_RESPONSE_V2 create context if the SMB2_DHANDLE_FLAG_PERSISTENT bit is not set in the **Flags** field of the request and if neither of the following conditions are met:

- **Open.OlockLevel** is equal to SMB2_OPLOCK_LEVEL_BATCH.
- **Open.Lease.LeaseState** has SMB2_LEASE_HANDLE_CACHING bit set.

The server MUST construct an SMB2_CREATE_DURABLE_HANDLE_RESPONSE_V2 response create context, with the following values set, as specified in section [2.2.14.2.12](#).

- If **Open.IsPersistent** is FALSE, **Open.DurableOpenTimeout** and the Timeout value in the response MUST be set to an implementation-specific value; <243> otherwise, the server MUST perform the following:
 - If the Timeout value in the request is not zero, the Timeout value in the response MUST be set to the Timeout value in the request.
 - If the Timeout value in the request is zero:
 - If **Share.CATimeout** is not zero, Timeout MUST be set to **Share.CATimeout**.
 - If **Share.CATimeout** is zero, Timeout SHOULD be set to an implementation-specific value. <244>

- **Open.DurableOpenTimeout** MUST be set to the Timeout value in the response.
- If **Open.IsPersistent** is TRUE, the server MUST set the SMB2_DHANDLE_FLAG_PERSISTENT bit in the Flags field.
- The buffer specified by the response MUST include **CreateContextLength** and **CreateContextOffset** fields.

3.3.5.9.11 Handling the SMB2_CREATE_REQUESTLEASE_V2 Create Context

This section applies only to servers that implement the SMB 3.0 dialect. If the name of the create context, as defined in section [2.2.13.2](#), is "RqLs" and the **DataLength** field equals 0x34, the server MUST process the create context as SMB2_CREATE_REQUESTLEASE_V2.

If both SMB2_CREATE_DURABLE_HANDLE_RECONNECT and SMB2_CREATE_REQUESTLEASE_V2 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 support directory leasing or **Connection.Dialect** is not equal to "3.000" or if the **RequestedOlockLevel** is not SMB2_OPLOCK_LEVEL_LEASE, the server MUST ignore the SMB2_CREATE_REQUESTLEASE_V2 Create Context request.

By specifying a **RequestedOlockLevel** 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_REQUESTLEASE_V2 Create Context, the lease request MUST be ignored and **Open.OlockLevel** 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_REQUESTLEASE_V2 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_REQUESTLEASE_V2 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.
- **Lease.Epoch** is set to 0.
- **Lease.Version** is set to 2.

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_REQUESTLEASE_V2 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 SMB2LEASE_WRITE_CACHING, the server MUST clear the bit SMB2LEASE_WRITE_CACHING in the **LeaseState** field.

If the **FileAttributes** field in the request indicates that this operation is on a directory and **LeaseState** includes SMB2LEASE_WRITE_CACHING, the server MUST clear the bit SMB2LEASE_WRITE_CACHING in the **LeaseState** field.

If **TreeConnect.Share.Type** includes STYPE_CLUSTER_SOFS, and if **LeaseState** includes SMB2LEASE_READ_CACHING, the server MUST set **LeaseState** to SMB2LEASE_READ_CACHING, otherwise set **LeaseState** to SMB2LEASE_NONE.

If the caching state requested in **LeaseState** of the SMB2_CREATE_REQUESTLEASE_V2 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. [<245>](#)

If the object store succeeds this request, **Lease.LeaseState** MUST be set to the new caching state. The server MUST increment **Lease.Epoch** by 1. If **Lease.Breaking** is TRUE, the server MUST return the existing **Lease.LeaseState** to client and set **Flags** to be SMB2LEASE_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_RESPONSELEASE_V2 response create context, following the syntax specified in section [2.2.14.2.11](#), 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**.

- If SMB2_LEASE_FLAG_PARENTLEASE_KEY_SET bit is set in the **Flags** field of the request, **ParentLeaseKey** MUST be set to the **ParentLeaseKey** in the request and SMB2_LEASE_FLAG_PARENTLEASE_KEY_SET bit MUST be set in the **Flags** field of the response.
- **Epoch** MUST be set to **Lease.Epoch**.

The server MUST set **Open.OlockState** to Held, set **Open.Lease** to a reference to Lease, set **Open.OlockLevel** to SMB2_OPLOCK_LEVEL_LEASE, and add Open to **Lease.LeaseOpens**. The remainder of open response construction continues as described in the "Response Construction" phase.

3.3.5.9.12 Handling the SMB2_CREATE_DURABLE_HANDLE_RECONNECT_V2 Create Context

This section applies only to servers that implement the SMB 3.0 dialect.

If the create request also contains SMB2_CREATE_DURABLE_HANDLE_REQUEST context or SMB2_CREATE_DURABLE_HANDLE_RECONNECT context or SMB2_CREATE_DURABLE_HANDLE_REQUEST_V2 context, the server MUST fail the request with STATUS_INVALID_PARAMETER.

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:

- The server MUST look up an existing open in the **GlobalOpenTable** by doing a lookup with the **FileId.Persistent** portion of the create context.
- 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](#).
- If **Open.ClientGuid** is not equal to the **ClientGuid** of the connection that received this request, the server MUST fail the create request with STATUS_OBJECT_NAME_NOT_FOUND.
- If **Open.FileName** does not match the file name specified in the **Buffer** field of the SMB2 CREATE request, the server SHOULD fail the request with STATUS_INVALID_PARAMETER.
- If **Open.IsPersistent** is TRUE and the SMB2_DHANDLE_FLAG_PERSISTENT bit is not set in the **Flags** field of the SMB2_CREATE_DURABLE_HANDLE_RECONNECT_V2 Create Context, the server MUST fail the request with STATUS_INVALID_PARAMETER.
- If **DesiredAccess** field of the SMB2 CREATE request is nonzero and does not match **Open.DesiredAccess**, the server MUST fail the request with STATUS_INVALID_PARAMETER.
- If **ShareAccess** field of the SMB2 CREATE request does not match **Open.ShareMode**, the server MUST fail the request with STATUS_INVALID_PARAMETER.
- If all the bits in the **CreateOptions** field of the SMB2 CREATE request, with the exception of FILE_COMPLETE_IF_OPLOCKED bit, do not match the corresponding bits in **Open.CreateOptions**, the server MUST fail the request with STATUS_INVALID_PARAMETER.
- If **Open.CreateGuid** is not equal to the **CreateGuid** in the request, the server MUST fail the request with STATUS_OBJECT_NAME_NOT_FOUND.

- 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](#).
- If **Open.Lease** is NULL, or if **Open.Session** is not NULL, the server MUST fail the request with STATUS_OBJECT_NAME_NOT_FOUND.
- If an SMB2_CREATE_REQUESTLEASE_V2 create context is also present in the request, the server supports directory leasing, and **Open.Lease.LeaseKey** does not match the **LeaseKey** provided in the SMB2_CREATE_REQUESTLEASE_V2 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](#).
- If an SMB2_CREATE_REQUESTLEASE create context is also present in the request, the server supports leasing, and **Open.Lease.LeaseKey** does not match the **LeaseKey** provided in the SMB2_CREATE_REQUESTLEASE 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](#).
- 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](#).
- The server MUST set the **Open.Connection** to refer to the connection that received this request.
- The server MUST set the **Open.Session** to refer to the session that received this request.
- 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.
- The server MUST regenerate **Open.FileId** (the volatile portion of the SMB2_FILEID).
- If the SMB2_DHANDLE_FLAG_PERSISTENT bit is set in the Flags field of the request and if **TreeConnect.Share.IsCA** is TRUE, the server MUST set **Open.IsPersistent** to TRUE.
- The server MUST insert the open into the **Session.OpenTable** with the **Open.FileId** as the new key.

The "Successful Open Initialization" and "Oplock Acquisition" phases MUST be skipped, and processing MUST continue as specified in "Response Construction".

In the "Response Construction" phase:

If the server supports directory leasing, and the request contains SMB2_CREATE_REQUESTLEASE_V2 Create Context, then the server MUST construct an SMB2_CREATE_RESPONSELEASE_V2 Create Context, following the syntax specified in section [2.2.14.2.10](#), and include it in the buffer described by the response **CreateContextLength** and **CreateContextOffset** fields. This structure MUST have the following values set:

- **LeaseKey** MUST be set to **Lease.LeaseKey**.
- **LeaseState** MUST be set to **Lease.LeaseState**.
- **ParentLeaseKey** MUST be set to the **ParentLeaseKey** in the request.
- The server MUST increment **Lease.Epoch** by 1. **Epoch** MUST be set to **Lease.Epoch**.

If the request contains an SMB2_CREATE_REQUESTLEASE Create Context, the server supports leasing and **Open.Lease** is not NULL, then the server MUST construct an SMB2_CREATE_RESPONSELEASE create context, following the syntax specified in section [2.2.14.2.10](#), and include it in the buffer described by the response **CreateContextLength** and **CreateContextOffset** fields. This structure MUST have the following values set:

- **LeaseKey** MUST be set to **Lease.LeaseKey**.
- **LeaseState** MUST be set to **Lease.LeaseState**.

3.3.5.9.13 Handling the SMB2_CREATE_APP_INSTANCE_ID Create Context

This section applies only to servers that implement the SMB 3.0 dialect.

The server MUST process this create context only after processing an SMB2_CREATE_DURABLE_HANDLE_REQUEST_V2 or SMB2_CREATE_DURABLE_HANDLE_RECONNECT_V2 create context in the request.

The server MUST attempt to locate an **Open** in **GlobalOpenTable** where:

- **AppInstanceId** in the request is equal to **Open.AppInstanceId**.
- Target path name is equal to **Open.PathName**.
- **Open.TreeConnect.Share** is equal to **Treeconnect.Share**.
- **Open.Session.Connection.ClientGuid** is not equal to the current **Connection.ClientGuid**.
- **Open.GrantedAccess** includes FILE_GENERIC_READ.

If an **Open** is found, the server MUST close the open as specified in [3.3.4.17](#).

If an **Open** is not found, the server MUST continue the create process specified in the "Open Execution" Phase. In the "Successful Open Initialization" phase, the server MUST set **Open.AppInstanceId** to the **AppInstanceId** in the create context request.

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, as specified in section [3.3.5.2.9](#).

The server MUST locate the tree connection, as specified in section [3.3.5.2.11](#).

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](#).

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.[246](#) This gives the client the attributes that have been updated to take into account any cached writes or extends that may have happened. The attributes that MUST be queried are the creation time, last access time, last write time, change time, allocation size in bytes, end of file in bytes, and file attributes.

The server then MUST construct the response following the syntax specified in section [2.2.16](#). The values MUST be set as follows:

- If the attributes of the file were requested and can be fetched, the server MUST set the **Flags** field to SMB2_CLOSE_FLAG_POSTQUERY_ATTRIB. Otherwise **Flags** MUST be set to 0.
- If SMB2_CLOSE_FLAG_POSTQUERY_ATTRIB was set:
 - **CreationTime**, **LastAccessTime**, **LastWriteTime**, **ChangeTime**, **AllocationSize**, **EndofFile**, and **FileAttributes** MUST be set to the values returned from the attribute query.
- If SMB2_CLOSE_FLAG_POSTQUERY_ATTRIB was not set:
 - **CreationTime**, **LastAccessTime**, **LastWriteTime**, **ChangeTime**, **AllocationSize**, **EndofFile**, and **FileAttributes** MUST all be set to 0.

The response MUST then be sent to the client.

The Server MUST send an [SMB2_CHANGE_NOTIFY Response](#) with STATUS_NOTIFY_CLEANUP status code for all pending CHANGE_NOTIFY requests associated with the **FileId** that is closed.

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, as specified in section [3.3.5.2.9](#).

The server MUST locate the tree connection, as specified in section [3.3.5.2.11](#).

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**.[<247>](#) 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<248> 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, as specified in section [3.3.5.2.9](#).

The server MUST locate the tree connection, as specified in section [3.3.5.2.11](#).

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<249> fail the request with STATUS_INVALID_PARAMETER.

If **Connection.SupportsMultiCredit** is TRUE the server MUST validate **CreditCharge** based on **Length**, as specified in section [3.3.5.2.5](#). If the validation fails, it MUST fail the read request with STATUS_INVALID_PARAMETER.

If **Connection.Dialect** is "3.000" and **Channel** is not equal to one of the values specified in section [2.2.19](#), the server MUST fail the request with STATUS_INVALID_PARAMETER.

If **Connection.Dialect** is "3.000" and **Channel** is equal to SMB2_CHANNEL_RDMA_V1 and any of the following conditions is TRUE, the server MUST fail the request with STATUS_INVALID_PARAMETER.

- Underlying **Connection** is not RDMA
- The **Length** or **ReadChannelInfoOffset** or **ReadChannelInfoLength** is equal to 0

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**.[<250>](#)

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[<251>](#) 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.[<252>](#) 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.

If the request **Channel** field does not contain the value SMB2_CHANNEL_RDMA_V1, then

- **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.

If the request **Channel** field contains the value SMB2_CHANNEL_RDMA_V1, the data MUST be sent via the processing specified in [\[MS-SMBD\]](#) section 3.1.4.5 RDMA Write to Peer Buffer, providing the **Connection**, the data, and the array of **SMB_DIRECT_BUFFER_DESCRIPTOR_V1** structures passed in the request at offset **ReadChannelInfoOffset** and of length **ReadChannelInfoLength** fields.

- The **DataOffset** field MUST be set to zero.
- The data MUST NOT 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, as specified in section [3.3.5.2.9](#).

The server MUST locate the tree connection, as specified in section [3.3.5.2.11](#).

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 the **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[<253>](#) 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[<254>](#) fail the request with STATUS_INVALID_PARAMETER.

If **Channel** is equal to 0 and **DataOffset** is less than 0x70, the server SHOULD[<255>](#) fail the request with STATUS_INVALID_PARAMETER.

If **Channel** is equal to 0 and **DataOffset** is greater than 0x100, the server MUST fail the request with STATUS_INVALID_PARAMETER.

If **Channel** is equal to 0 and the number of bytes received in **Buffer** is less than (**DataOffset** + **Length**), the server MUST fail the request with STATUS_INVALID_PARAMETER.

If **Connection.SupportsMultiCredit** is TRUE, the server MUST validate **CreditCharge** based on **Length**, as specified in section [3.3.5.2.5](#). If the validation fails, it MUST fail the write request with STATUS_INVALID_PARAMETER.

If **Connection.Dialect** is "3.000" and **Channel** is not equal to one of the values specified in section [2.2.19](#), the server MUST fail the request with STATUS_INVALID_PARAMETER.

If **Connection.Dialect** is "3.000" and **Channel** is equal to SMB2_CHANNEL_RDMA_V1 and any of the following conditions is TRUE, the server MUST fail the request with STATUS_INVALID_PARAMETER.

- Underlying **Connection** is not RDMA
- **RemainingBytes** is equal to 0
- **Length** or **DataOffset** is not equal to 0
- **WriteChannelInfoOffset** or **WriteChannelInfoLength** is equal to 0

If the request **Channel** field contains the value SMB2_CHANNEL_RDMA_V1, then the data MUST be first obtained via the processing specified in [\[MS-SMBD\]](#) section 3.1.4.5 RDMA Read from Peer Buffer, providing the **Connection**, a newly allocated buffer to receive the data, and the array of SMB_DIRECT_BUFFER_DESCRIPTOR_V1 structures passed in the request at offset **WriteChannelInfoOffset** and of length **WriteChannelInfoLength** fields.

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 **Connection.Dialect** is not "2.002", and SMB2_WRITEFLAG_WRITE_THROUGH is set in the **Flags** field of the SMB2 WRITE Request, the server SHOULD [<256>](#) 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 [<257>](#) 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 [\[MS-FSCI\]](#) 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.

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 that has an [SMB2 header \(section 2.2.1\)](#) with a **Command** value equal to SMB2 LOCK, message handling proceeds as follows:

The server MUST locate the Session, as specified in section [3.3.5.2.9](#).

The server MUST locate the Tree Connect, as specified in section [3.3.5.2.11](#).

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 **LockSequence** value in the [SMB2 LOCK Request \(section 2.2.26\)](#) is not zero, and either one of the following conditions is TRUE, the server SHOULD verify whether the lock/unlock request with that **LockSequence** value has been successfully processed before.

- **Connection.Dialect** is "2.100" and **Open.IsResilient** is TRUE.
- **Connection.Dialect** is "3.000".[<258>](#)

The server verifies the **LockSequence** by performing the following steps:

1. The server MUST compute ((**LockSequence** >> 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 it 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 set the entry value to 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 [<259>](#) 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. [<260>](#) 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, **Connection.Dialect** is "2.100" or "3.000", 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 ((**LockSequence** >> 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^{<261>} fail the request with STATUS_INVALID_PARAMETER. For each SMB2_LOCK_ELEMENT entry in the **Locks** array, if SMB2_LOCKFLAG_UNLOCK is set, the server MUST fail the request with STATUS_INVALID_PARAMETER and stop processing further entries in the **Locks** array. All successfully processed Lock operations are not rolled back. 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.^{<262>} 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.^{<263>}

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, **Connection.Dialect** is "2.100" or "3.000", 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 ((**LockSequence** >> 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, as specified in section [3.3.5.2.9](#).

The server MUST locate the tree connection, as specified in section [3.3.5.2.11](#).

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 **Connection.Dialect** is "3.000" and if the operation represented by the **CtlCode** in the request is not allowed on the server, the server SHOULD[<264>](#) fail the request with STATUS_NOT_SUPPORTED.

If the **CtlCode** is FSCTL_DFS_GET_REFERRALS, FSCTL_DFS_GET_REFERRALS_EX, FSCTL_QUERY_NETWORK_INTERFACE_INFO, FSCTL_VALIDATE_NEGOTIATE_INFO, or FSCTL_PIPE_WAIT and the value of **FileId** in the SMB2 Header of the request is not 0xFFFFFFFFFFFFFF, then the server MUST fail the request with STATUS_INVALID_PARAMETER.

For **CtlCode** values other than FSCTL_DFS_GET_REFERRALS, FSCTL_DFS_GET_REFERRALS_EX, FSCTL_QUERY_NETWORK_INTERFACE_INFO, FSCTL_VALIDATE_NEGOTIATE_INFO, 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**.

If either **InputCount**, **MaxInputResponse**, or **MaxOutputResponse** is greater than an implementation-specific value, the server SHOULD[<265>](#) fail the request with STATUS_INVALID_PARAMETER.

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^{<266>} 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 3.3.5.2.5. If the validation fails, it MUST fail the IOCTL request with STATUS_INVALID_PARAMETER.

A server implementing the SMB 2.002 or SMB 2.1 dialect SHOULD^{<267>} 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.[<268>](#) 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[<269>](#) 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 server 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 or FSCTL_DFS_GET_REFERRALS_EX, 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 invoke the event as specified in [\[MS-DFSC\]](#) section 3.2.4.2 and pass the following:

- The IP address of the client.
- The buffer containing the DFS referral request packet.
- **IsExtendedReferral**: Set to TRUE when **CtlCode** is FSCTL_DFS_GET_REFERRALS_EX.
- The maximum size of the response data buffer that will be accepted by the client, as indicated by **MaxOutputResponse** field in the request.

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^{<270>} 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 the **CtlCode** in the request.
- **FileId** MUST be set to { 0xFFFFFFFFFFFFFF, 0xFFFFFFFFFFFFFF }.
- **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^{<271>} 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, 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 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 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 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 [<276>](#) 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.<277>](#)

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.[<278>](#)

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.[^{<279>}](#) If the **SourceOffset** or **SourceOffset + Length** extends beyond the end of file, the server SHOULD[^{<280>}](#) 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.[^{<281>}](#) 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 that has 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 \(section 2.2.31.2\)](#) 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.[<282>](#)
- 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.
- The server MUST fail the SRV_READ_HASH request with an error of STATUS_INVALID_PARAMETER in the following cases:
 - If the **HashType** field of the SRV_READ_HASH request is not equal to SRV_HASH_TYPE_PEER_DIST.
 - If the server implements only the SMB 2.1 dialect and the **HashVersion** field is not equal to SRV_HASH_VER_1.
 - If the server implements the SMB 3.0 dialect and the **HashVersion** field is not equal to either SRV_HASH_VER_1 or SRV_HASH_VER_2.
 - If the **HashRetrievalType** field is not equal to SRV_HASH_RETRIEVE_HASH_BASED or SRV_HASH_RETRIEVE_FILE_BASED.
 - If the **HashVersion** field is equal to SRV_HASH_VER_1 and the **HashRetrievalType** field is not equal to SRV_HASH_RETRIEVE_HASH_BASED.
 - If the **HashVersion** field is equal to SRV_HASH_VER_2 and the **HashRetrievalType** field is not equal to SRV_HASH_RETRIEVE_FILE_BASED.
- If **ServerHashLevel** is **HashDisableAll**, the server MUST fail the SRV_READ_HASH request with error code STATUS_HASH_NOT_SUPPORTED.
- If the **HashRetrievalType** is SRV_HASH_RETRIEVE_HASH_BASED the server MUST open the Content Information File from the object store for the object represented by **Open.LocalOpen**

with the specified offset. If the Content Information File open fails, the server MUST fail the request with STATUS_HASH_NOT_PRESENT.

- If the **HashRetrievalType** is SRV_HASH_RETRIEVE_FILE_BASED the server MUST open the Content Information File from the object store for the object represented by **Open.LocalOpen**. If the Content Information File open fails, the server MUST fail the request with STATUS_HASH_NOT_PRESENT.
- 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 **HashRetrievalType** is SRV_HASH_RETRIEVE_HASH_BASED and 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.
- If the **HashRetrievalType** is SRV_HASH_RETRIEVE_FILE_BASED and **Offset** field of the SRV_READ_HASH request is equal to or beyond the end of the file represented by **Open.LocalOpen**, 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 [2.2.32.4.1](#).
 - 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 server implements the SMB 3.0 dialect and the **HashVersion** field in the SRV_READ_HASH Request is SRV_HASH_VER_2, the server MUST set **HashBlobLength** in the HASH_HEADER to zero.

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.
- If the **HashRetrievalType** is SRV_HASH_RETRIEVE_HASH_BASED, the server MUST copy a SRV_READ_HASH Response following the syntax specified in section [2.2.32.4.2](#) into the **Buffer** field at the **OutputOffset** computed above.
- If the **HashRetrievalType** is SRV_HASH_RETRIEVE_FILE_BASED, the server MUST copy a SRV_READ_HASH Response following the syntax specified in section [2.2.32.4.3](#) into the **Buffer** field at the **OutputOffset** computed above. If the **Offset** field in the SRV_READ_HASH request is zero, the server MUST also copy the HASH_HEADER from the Content Information File, as specified in section [2.2.32.4.1](#), at the beginning of the **Buffer[]** field of the response.

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 [<283>](#) 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 [<284>](#) 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 [<285>](#) 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

This section applies only to servers that implement the SMB 2.1 or the SMB 3.0 dialect.

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 **Open.Connection.Dialect** is "2.002", the server SHOULD [fail the request with STATUS_INVALID_DEVICE_REQUEST](#).

Otherwise, if the server does not support FSCTL_LMR_REQUEST_RESILIENCY requests, the server SHOULD fail the request with **STATUS_NOT_SUPPORTED**.

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. If the value of the **Timeout** field specified in **NETWORK_RESILIENCY_REQUEST** of the request is not zero, **Open.ResiliencyTimeout** MUST be set to the value of the **Timeout** field; otherwise, **Open.ResiliencyTimeout** SHOULD be set to an implementation-specific value. [<287>](#) **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 { 0xFFFFFFFFFFFFFF, 0xFFFFFFFFFFFFFF }.
- The **OutputCount** field MUST be set to 0.

The response MUST be sent to the client.

3.3.5.15.11 Handling a Query Network Interface Request

When the server receives a request with an SMB2 header with a **Command** value equal to SMB2 IOCTL and a **CtlCode** of FSCTL_QUERY_NETWORK_INTERFACE_INFO, message handling proceeds as follows:

The server MUST query the registered transport interface, as specified in section [2.1](#) and construct the NETWORK_INTERFACE_INFO structure as specified in section [2.2.32.5](#).

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_QUERY_NETWORK_INTERFACE_INFO.
- The **FileId** field MUST be set to { 0xFFFFFFFFFFFFFF, 0xFFFFFFFFFFFFFF }.
- **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 NETWORK_INTERFACE_INFO that was previously constructed.
- **Flags** MUST be set to zero.
- The server MUST copy the constructed NETWORK_INTERFACE_INFO into the **Buffer** field at the **OutputOffset** that was previously computed.

The response MUST be sent to the client.

3.3.5.15.12 Handling a Validate Negotiate Info Request

When the server receives a request with an SMB2 header with a **Command** value equal to SMB2 IOCTL, and a **CtlCode** of FSCTL_VALIDATE_NEGOTIATE_INFO, message handling proceeds as follows:

- If **MaxOutputResponse** in the request is less than the size of a VALIDATE_NEGOTIATE_INFO Response, the server MUST fail the request with error STATUS_BUFFER_TOO_SMALL.
- The server MUST determine the greatest common dialect between the dialects it implements and the **Dialects** array of the VALIDATE_NEGOTIATE_INFO request. If no dialect is matched, or if the value is not equal to **Connection.Dialect**, the server MUST fail the request with STATUS_ACCESS_DENIED, terminate the transport connection, and free the **Connection** object.
- If the **Guid** received in the VALIDATE_NEGOTIATE_INFO request structure is not equal to the **Connection.ClientGuid**, the server MUST fail the request with STATUS_ACCESS_DENIED, terminate the transport connection, and free the **Connection** object.
- If the **SecurityMode** received in the VALIDATE_NEGOTIATE_INFO request structure is not equal to **Connection.ClientSecurityMode**, the server MUST fail the request with STATUS_ACCESS_DENIED, terminate the transport connection, and free the **Connection** object.
- If **Connection.ClientCapabilities** is not equal to the **Capabilities** received in the VALIDATE_NEGOTIATE_INFO request structure, the server MUST fail the request with the error STATUS_ACCESS_DENIED, terminate the transport connection, and free the **Connection** object.

The server MUST construct the VALIDATE_NEGOTIATE_INFO Response specified in section [2.2.32.6](#), as follows:

- **Capabilities** is set to **Connection.ServerCapabilities**.
- **Guid** is set to **ServerGuid**.
- **SecurityMode** is set to **Connection.ServerSecurityMode**.
- **Dialect** is set to **Connection.Dialect**.

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_VALIDATE_NEGOTIATE_INFO.
- **FileId** MUST be set to { 0xFFFFFFFFFFFFFF, 0xFFFFFFFFFFFFFF }.

- **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 VALIDATE_NEGOTIATE_INFO response that is constructed as above.
- **Flags** MUST be set to zero.
- The server MUST copy the constructed VALIDATE_NEGOTIATE_INFO Response structure into the **Buffer** field at the **OutputOffset** computed above.

The response MUST be sent to the client.

3.3.5.15.13 Handling a Set Reparse Point Request

This section applies only to servers that implement the SMB 3.0 dialect.

When the server receives a request that contains an SMB2 header with a **Command** value equal to SMB2 IOCTL and a **CtlCode** of FSCTL_SET_REPARSE_POINT, message handling proceeds as follows:

If the **ReparseTag** field in FSCTL_SET_REPARSE_POINT, as specified in [MS-FSCC] section 2.3.57, is not IO_REPARSE_TAG_SYMLINK, the server SHOULD verify that the caller has the required permissions to execute this FSCTL.<288> If the caller does not have the required permissions, the server MUST fail the call with an error code of STATUS_ACCESS_DENIED.

The server MUST process this request as a pass-through operation as specified in section [3.3.5.15.8](#).

3.3.5.15.14 Handling a File Level Trim Request

This section applies only to servers that implement the SMB 3.0 dialect.

When the server receives a request that contains an SMB2 header with a **Command** value equal to SMB2 IOCTL and a **CtlCode** of FSCTL_FILE_LEVEL_TRIM, message handling proceeds as follows:

If the **Key** field in FSCTL_FILE_LEVEL_TRIM, as specified in [MS-FSCC] section 2.3.69, is not zero, the server MUST fail the request with an error code of STATUS_INVALID_PARAMETER.

The server MUST process this request as a pass-through operation as specified in section [3.3.5.15.8](#).

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 [3.3.5.2.3](#) and [3.3.5.2.4](#).

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.[289](#) 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, as specified in section [3.3.5.2.9](#).

The server MUST locate the tree connection, as specified in section [3.3.5.2.11](#).

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 [3.3.5.2.5](#). 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[<290>](#) 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[<291>](#) 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[<292>](#) 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**.[<293>](#) 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, as specified in section [3.3.5.2.9](#).

The server MUST locate the tree connection, as specified in section [3.3.5.2.11](#).

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**[**<294>**](#), 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 [3.3.5.2.5](#). 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[**<295>**](#) 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.[**<296>**](#) 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, as specified in section [3.3.5.2.9](#).

The server MUST locate the tree connection, as specified in section [3.3.5.2.11](#).

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 [3.3.5.2.5](#). If the validation fails, it MUST fail the request with STATUS_INVALID_PARAMETER.

The server MUST verify the **InputBufferLength** as noted in the following:

- For quota requests, if the **InputBufferLength** is not equal to the size of SMB2_QUERY_QUOTA_INFO in the request, the server MUST fail the request with STATUS_INVALID_PARAMETER.
- For FileFullEaInformation requests, if **InputBufferLength** is not equal to the size of **Buffer** in the request, the server MUST fail the request with STATUS_INVALID_PARAMETER.
- For other information queries, the server MUST ignore the **InputBufferLength** value.

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 [`<297>`](#) be failed with STATUS_INVALID_INFO_CLASS.

If the request is for the FilePositionInformation information class, the SMB2 server SHOULD [`<298>`](#) 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. [`<299>`](#) 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.[<300>](#) 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 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.[<301>](#)

3.3.5.20.3 Handling SMB2_0_INFO_SECURITY

This section assumes knowledge about security concepts, as described in [\[MS-WPO\]](#) section 8 and specified in [\[MS-DTYP\]](#).

The server SHOULD[<302>](#) call into the underlying object store to query the security descriptor for the object.

The fields required in the resulting security descriptor are denoted by the flags given in the **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.[<303>](#)

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.[<304>](#)

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_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, as specified in section [3.3.5.2.9](#).

The server MUST locate the tree connection, as specified in section [3.3.5.2.11](#).

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 the **BufferLength** field is zero, the server SHOULD fail the request with STATUS_INVALID_PARAMETER.

If **Connection.SupportsMultiCredit** is TRUE, the server MUST validate **CreditCharge** based on **BufferLength**, as specified in section [3.3.5.2.5](#). 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 [^{<305>}](#) 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. [^{<306>}](#) 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, **Connection.Dialect** is "2.100" or "3.000", the server 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.[<307>](#) 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 [\[MS-WPO\]](#) section 8 and specified in [\[MS-DTYP\]](#).[<308>](#)

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.[<309>](#)

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](#) following the syntax given in section [2.2.40](#).

The response MUST then be sent to the client.

3.3.5.21.4 Handling SMB2_0_INFO_QUOTA

The server's object store MAY support quotas associated with a security principal. If the server exposes support for quotas, it MUST allow security principals to be identified using security identifiers (SIDs) in the format specified in [\[MS-DTYP\]](#) section 2.4.2.2.[<310>](#)

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.[311](#)

If the underlying object store returns an error, the server MUST fail the request with the error code received.

Otherwise, the server MUST initialize an [SMB2_SET_INFO Response](#) following the syntax given in section [2.2.40](#).

The response MUST then be sent to the client.

3.3.5.22 Receiving an SMB2_OPLOCK_BREAK Acknowledgment

When the server receives a request with an [SMB2 header](#) with a **Command** value equal to SMB2_OPLOCK_BREAK, message handling proceeds as follows:

- If **Connection.Dialect** is equal to "2.100" or "3.000" 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](#).
- Otherwise, the server MUST process the request as described in section [3.3.5.22.1](#).

3.3.5.22.1 Processing an Olock Acknowledgment

The server MUST locate the session, as specified in section [3.3.5.2.9](#).

The server MUST locate the tree connection, as specified in section [3.3.5.2.11](#).

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.OlockState** is not Breaking, the server MUST fail the request with STATUS_INVALID_DEVICE_STATE.

If the **OlockLevel** 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.OlockLevel** is not SMB2_OPLOCK_LEVEL_EXCLUSIVE or SMB2_OPLOCK_LEVEL_BATCH, and the **OlockLevel** 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 **OlockLevel** of the request is SMB2_OPLOCK_LEVEL_II, the server MUST set **Open.OlockLevel** to SMB2_OPLOCK_LEVEL_II and **Open.OlockState** to Held. If the **OlockLevel** of the request is SMB2_OPLOCK_LEVEL_NONE, the server MUST set **Open.OlockLevel** to SMB2_OPLOCK_LEVEL_NONE and **Open.OlockState** to None.

The server then MUST construct an oplock break response using the syntax specified in section [2.2.25](#) with the following value:

- **OlockLevel** MUST be set to **Open.OlockLevel**.

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, as specified in section [3.3.5.2.9](#).

The server MUST locate the tree connection, as specified in section [3.3.5.2.11](#).

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_OBJECT_NAME_NOT_FOUND.

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_OBJECT_NAME_NOT_FOUND.

If **Lease.Breaking** is FALSE, the server MUST fail the request with STATUS_UNSUCCESSFUL.

If **LeaseState** is not <= **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 a lease 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_OBJECT_NAME_NOT_FOUND
- 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 **Open.Connection.Dialect** is "2.100" or "3.000", and the server 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 the SMB 2.1 or SMB 3.0 dialect 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, for each session in **Connection.SessionTable**, the server MUST perform the following:

If **Connection.Dialect** is "3.000" and if the **Session** has more than one channel in **Session.ChannelList**, the server MUST perform the following action:

- All requests in **Session.Channel.Connection.RequestList** MUST be canceled. The server SHOULD pass the **CancelRequestId** to the object store to request cancellation of the pending operation.
- The channel entry MUST be removed from the **Session.ChannelList** where **Channel.Connection** matches the disconnected connection.

Otherwise, the server MUST perform the following actions:

- 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.
 - **Open.Connection.Dialect** is "2.100" or "3.000", the server supports leasing and **Open.IsResilient** is TRUE.
 - **Open.OlockLevel** is equal to SMB2_OPLOCK_LEVEL_BATCH and **Open.OlockState** is equal to Held, and **Open.IsDurable** is TRUE.
 - **Open.OlockLevel** is equal to SMB2_OPLOCK_LEVELLEASE, **Lease.LeaseState** contains SMB2LEASE_HANDLE_CACHING, **Open.OlockState** 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, the server MUST do the following:
 - If **Open.Connection.Dialect** is "3.000", and if **Open.DurableOpenTimeOut** is not zero, the server MUST add the current time to its value.
 - Otherwise, the server MUST set **Open.DurableOpenTimeOut** to the current time plus an implementation-specific default value. [<313>](#)
 - The server MUST start the durable open scavenger timer, as specified in sections [3.3.2.2](#).

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 [<314>](#) 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.

4 Protocol Examples

The following sections describe common scenarios that indicate normal traffic flow in order to illustrate the function of the SMB 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.

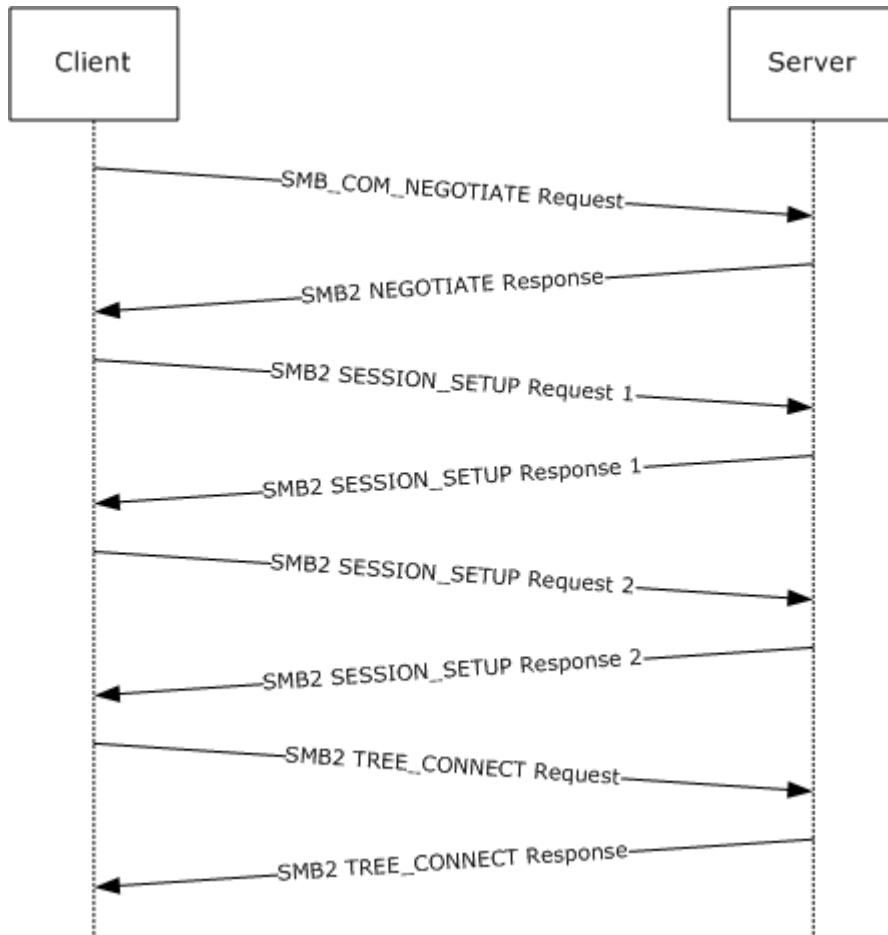


Figure 6: Client negotiating SMB2 with SMB-style negotiate

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 implements.

```
Smb: C; Negotiate, Dialect = PC NETWORK PROGRAM 1.0, LANMAN1.0, Windows for Workgroups  
3.1a, LM1.2X002, LANMAN2.1, NT LM 0.12, SMB 2.002  
Protocol: SMB  
Command: Negotiate 114(0x72)  
SMBHeader: Command, TID: 0xFFFF, PID: 0xFEFF, UID: 0x0000, MID: 0x0000  
Flags: 24 (0x18)
```

```

Bit0: (.....0) SMB_FLAGS_LOCK_AND_READ_OK: LOCK_AND_READ and WRITE_AND_CLOSE not
supported (obsoleted)
Bit1: (.....0.) SMB_FLAGS_SEND_NO_ACK [not implemented]
Bit2: (....0..) Reserved (must be zero)
Bit3: (....1...) SMB_FLAGS_CASE_INSENSITIVE: SMB paths are case-insensitive
Bit4: (...1....) SMB_FLAGS_CANONICALIZED_PATHS: Canonicalized File and pathnames
(obsoleted)
Bit5: (.0.....) SMB_FLAGS_OPLOCK: No Olocks supported for OPEN, CREATE & CREATE_NEW
(obsoleted)
Bit6: (.0.....) SMB_FLAGS_OPLOCK_NOTIFY_ANY: No Notifications supported for OPEN,
CREATE & CREATE_NEW (obsoleted)
Bit7: (0.....) SMB_FLAGS_SERVER_TO_REDIR: Command - SMB is being sent from the
client
Flags2: 51283 (0xC853)
Bit00: (.....1) SMB_FLAGS2_KNOWS_LONG_NAMES: May return long file names
Bit01: (.....1.) SMB_FLAGS2_KNOWS_EAS: Understands extended attributes
Bit02: (.....0..) SMB_FLAGS2_SMB_SECURITY_SIGNATURE: Not security signature-
enabled
Bit03: (.....0...) Reserved
Bit04: (.....1....) Reserved
Bit05: (.....0....) SMB_FLAGS2_SMB_SECURITY_SIGNATURE_REQUIRED: SMB packets must
be signed
Bit06: (.....1.....) SMB_FLAGS2_IS_LONG_NAME: Any path name in the request is a
long name
Bit07: (.....0.....) Reserved
Bit08: (.....0.....) Reserved
Bit09: (.....0.....) Reserved
Bit10: (.....0.....) SMB_FLAGS2_REPARSE_PATH: Not requesting Reparse path
Bit11: (....1.....) SMB_FLAGS2_EXTENDED_SECURITY: Aware of extended security
Bit12: (....0.....) SMB_FLAGS2_DFS: No DFS namespace
Bit13: (....0.....) SMB_FLAGS2_PAGING_IO: Read operation will NOT be permitted
if has no read permission
Bit14: (.1.....) SMB_FLAGS2_NT_STATUS: Using 32-bit NT status error codes
Bit15: (1.....) SMB_FLAGS2_UNICODE: Using UNICODE strings
PIDHigh: 0 (0x0)
SecuritySignature: 0x0
Reserved: 0 (0x0)
TreeID: 65535 (0xFFFF)
Reserved: 0 (0x0)
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)
CreditCharge: 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)
Reserved: 0 (0x0)
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)
CreditCharge: 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)
Reserved: 0 (0x0)
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)
CreditCharge: 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)
Reserved: 0 (0x0)
TreeId: 0 (0x0)
SessionId: 439804651113 (0x400000000009)
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)
CreditCharge: 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)
Reserved: 0 (0x0)
TreeId: 0 (0x0)
SessionId: 439804651113 (0x400000000009)
CSessionSetup:
Size: 25 (0x19)
VcNumber: 0 (0x0)
SecurityMode: Signing Enabled
Capabilities: 1 (0x1)
DFS: .....1 DFS available
Channel: 0 (0x0)
SecurityBufferOffset: 88 (0x58)
SecurityBufferLength: 245 (0xF5)
Buffer: (245 bytes)
```

6. The server processes the token received with GSS and gets a successful return code. The server responds to client with an SMB2 SESSION_SETUP Response with **Status** equal to STATUS_SUCCESS and the response containing the output token from GSS.

```
Smb2: R SESSION SETUP
Smb2: R SESSION SETUP
SMB2Header:
Size: 64 (0x40)
CreditCharge: 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)
Reserved: 0 (0x0)
TreeId: 0 (0x0)
SessionId: 439804651113 (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)
CreditCharge: 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)
Reserved: 0 (0x0)
TreeId: 0 (0x0)
SessionId: 439804651113 (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)
CreditCharge: 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)
Reserved: 0 (0x0)
TreeId: 1 (0x1)
SessionId: 4398046511113 (0x400000000009)
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.

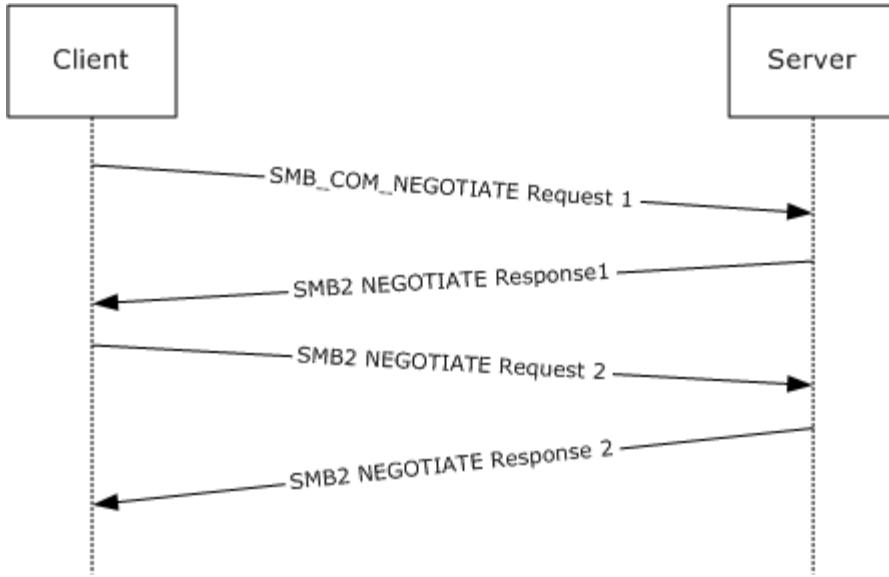


Figure 7: Client negotiating SMB 2.10 dialect with SMB-style negotiate

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 implements.

```

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_REDIRECT)
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)
Reserved:        0 (0x0)
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)
CreditCharge: 0 (0x0)
Status: 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
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: (....000000000000000000000000....)
DFS: (....0.....Command is not a DFS Operation
(SMB2_FLAGS_DFS_OPERATIONS)
Reserved29_31: (000.....)
NextCommand: 0 (0x0)
MessageId: 0 (0x0)
Reserved: 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
```

```

SMBIdentifier: SMB
SMB2Header: C NEGOTIATE (0x0)
Size: 64 (0x40)
CreditCharge: 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: 0 (0x0)
Flags: 0x0
ServerToRedir: (.....0) Client to Server
(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: (....000000000000000000000000....)
DFS: (....0.....) Command is not a DFS Operation
(SMB2_FLAGS_DFS_OPERATIONS)
Reserved29_31: (000.....)
NextCommand: 0 (0x0)
MessageId: 1 (0x1)
Reserved: 0 (0x0)
TreeId: 0 (0x0)
SessionId: 0 (0x0)
Signature: Binary Large Object (16 Bytes)
CNegotiate:
Size: 36 (0x24)
DialectCount: 2 (0x2)
SecurityMode: Signing Enabled (0x1)
Reserved: 0 (0x0)
Capabilities: 0x0
DFS: (.....0) DFS unavailable
Reserved_bits1_31: (00000000000000000000000000000000.) Reserved
Guid: {9879BE56-0D00-58BA-11DD-D5F0AF3A5B5D}
StartTime: No Time Specified (0)
Dialects:
Dialects: 514 (0x202)
Dialects: 528 (0x210)

```

4. The server receives the SMB2 negotiate request and finds dialect 0x0210. The server sends an SMB2 NEGOTIATE Response with DialectRevision set to 0x0210.

```

Smb2: R NEGOTIATE (0x0), GUID={1ED9580F-5FEF-1AA0-4B9D-DB1C77C63757}, Mid = 1
SMBIdentifier: SMB
SMB2Header: R NEGOTIATE (0x0)
Size: 64 (0x40)
CreditCharge: 0 (0x0)
Status: 0x0, Facility = FACILITY_SYSTEM, Severity = STATUS_SEVERITY_SUCCESS, Code =
(0) STATUS_SUCCESS
Code: (.....0000000000000000) (0) STATUS_SUCCESS

```

```

Facility: (...00000000000000...) 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: (...000000000000000000000000....)
DFS: (...0.....) Command is not a DFS Operation
(SMB2_FLAGS_DFS_OPERATIONS)
Reserved29_31: (000.....)
NextCommand: 0 (0x0)
MessageId: 1 (0x1)
Reserved: 0 (0x0)
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.

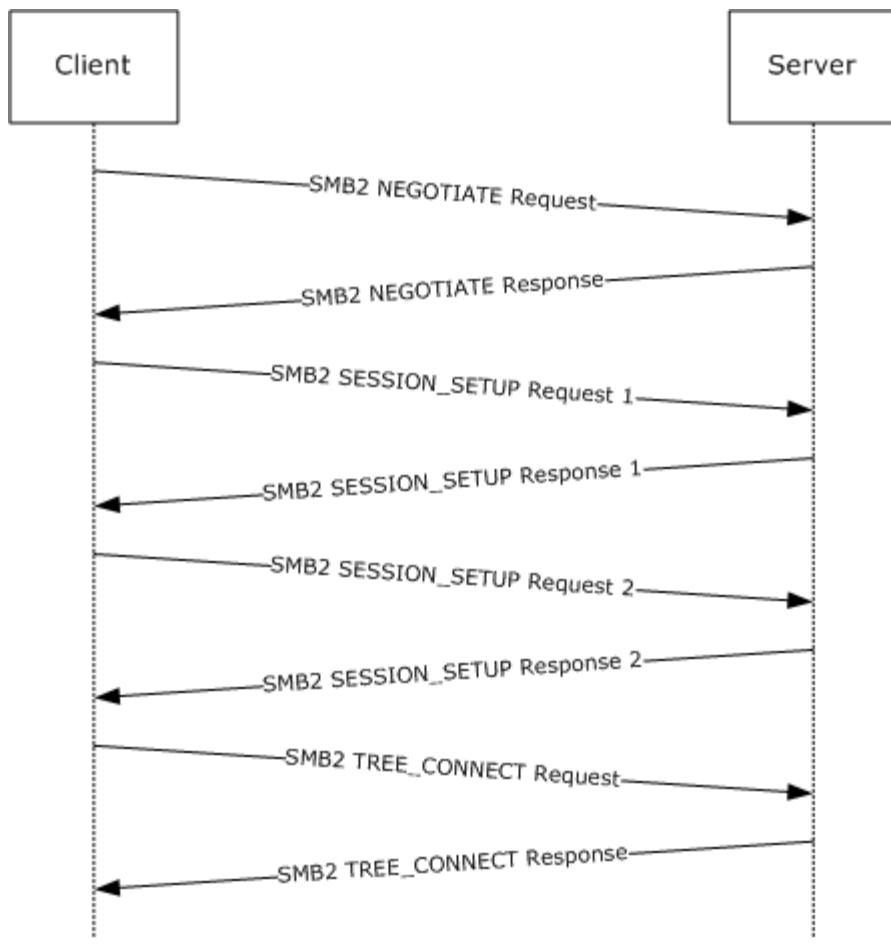


Figure 8: Client negotiating SMB2 with SMB2 negotiate

1. The client sends an SMB2 negotiate packet with the dialect 0x0202 in the **Dialects** array.

```

Smb2: C NEGOTIATE
SMB2Header:
Size: 64 (0x40)
CreditCharge: 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)
Reserved: 0 (0x0)
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)
CreditCharge: 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)
Reserved: 0 (0x0)
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)
CreditCharge: 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)
Reserved: 0 (0x0)
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)
CreditCharge: 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)
Reserved: 0 (0x0)
TreeId: 0 (0x0)
SessionId: 439804651113 (0x400000000009)
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)
CreditCharge: 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)
Reserved: 0 (0x0)
TreeId: 0 (0x0)
SessionId: 4398046511113 (0x400000000009)
CSessionSetup:
Size: 25 (0x19)
VcNumber: 0 (0x0)
SecurityMode: Signing Enabled
Capabilities: 1 (0x1)
DFS: .....1 DFS available
Channel: 0 (0x0)
SecurityBufferOffset: 88 (0x58)
SecurityBufferLength: 245 (0xF5)
Buffer: (245 bytes)
```

6. The server processes the token received with GSS and gets a successful return code. The server responds to 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)
CreditCharge: 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)
Reserved: 0 (0x0)
TreeId: 0 (0x0)
SessionId: 439804651113 (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)
CreditCharge: 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)
Reserved: 0 (0x0)
TreeId: 0 (0x0)
SessionId: 439804651113 (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)
CreditCharge: 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)
Reserved: 0 (0x0)
TreeId: 1 (0x1)
SessionId: 439804651113 (0x400000000009)
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** = 0x400000000D and **TreeId** 0x1, and messages have been exchanged such that the current **MessageId** is 9.

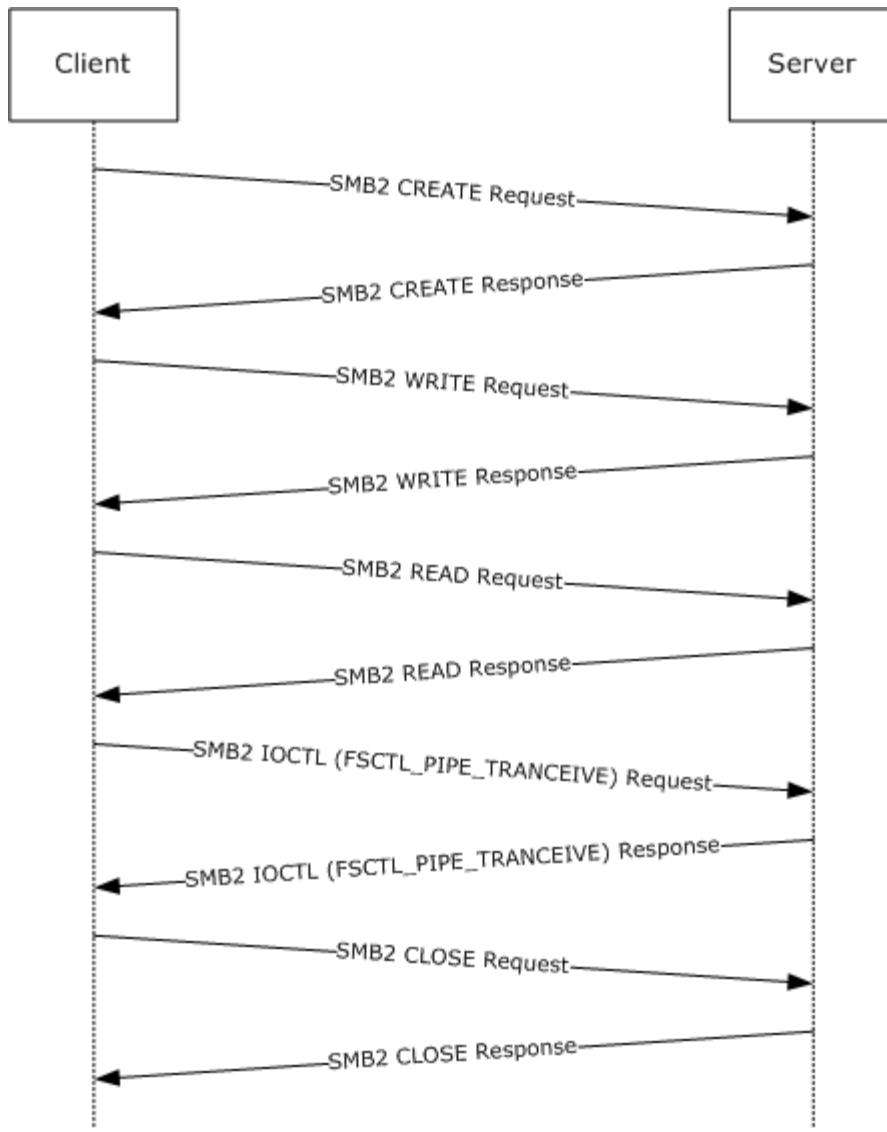


Figure 9: Executing an operation on a named pipe

1. The client sends an [SMB2 CREATE Request](#) to open the named pipe "srvsvc".

```

Smb2: C CREATE srvsvc
SMB2Header:
Size: 64 (0x40)
CreditCharge: 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)
Reserved: 0 (0x0)
TreeId: 1 (0x1)
SessionId: 439804651117 (0x400000000D)
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)
CreditCharge: 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)
Reserved: 0 (0x0)
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)
CreditCharge: 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)
Reserved: 0 (0x0)
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)
CreditCharge: 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)
Reserved: 0 (0x0)
TreeId: 1 (0x1)
SessionId: 439804651117 (0x400000000D)
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)
CreditCharge: 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)
Reserved: 0 (0x0)
TreeId: 1 (0x1)
SessionId: 439804651117 (0x400000000D)
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)
CreditCharge: 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)
Reserved: 0 (0x0)
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)
CreditCharge: 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)
Reserved: 0 (0x0)
TreeId: 1 (0x1)
SessionId: 4398046511117 (0x4000000000D)
CIoCtl:
Size: 57 (0x39)
Reserved: 0 (0x0)
```

```

Code: 0x0011c017
Fid:
Persistent: 5 (0x5)
Volatile: -4294967291 (0xFFFFFFFF00000005)
InputOffset: 120 (0x78)
InputCount: 68 (0x44)
MaxInputResponse: 0 (0x0)
OutputOffset: 120 (0x78)
OutputCount: 0 (0x0)
MaxOutputResponse: 1024 (0x400)
Flags: 1 (0x1)
Reserved2: 0 (0x0)
Input: (68 bytes)

```

8. The server sends an [SMB2 IOCTL Response](#) with the data that was read.

```

Smb2: R IOCTL
SMB2Header:
Size: 64 (0x40)
CreditCharge: 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)
Reserved: 0 (0x0)
TreeId: 1 (0x1)
SessionId: 439804651117 (0x400000000D)
RIoCtl:
Size: 49 (0x31)
Reserved: 0 (0x0)
Code: 0x0011c017
Method: .....11 Method neither
Function: 0x005
Access: .....11..... Read/Write
Device: 0x0011
Fid:
Persistent: 5 (0x5)
Volatile: -4294967291 (0xFFFFFFFF00000005)
InputOffset: 112 (0x70)
InputCount: 68 (0x44)
OutputOffset: 184 (0xB8)
OutputCount: 112 (0x70)
Flags: 0 (0x0)
Reserved2: 0 (0x0)
Input: (68 bytes)
Output: (112 bytes)

```

9. The client sends an [SMB2 CLOSE Request](#) to close the named pipe.

```
Smb2: C CLOSE FID=
SMB2Header:
Size: 64 (0x40)
CreditCharge: 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)
Reserved: 0 (0x0)
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)
CreditCharge: 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)
Reserved: 0 (0x0)
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.

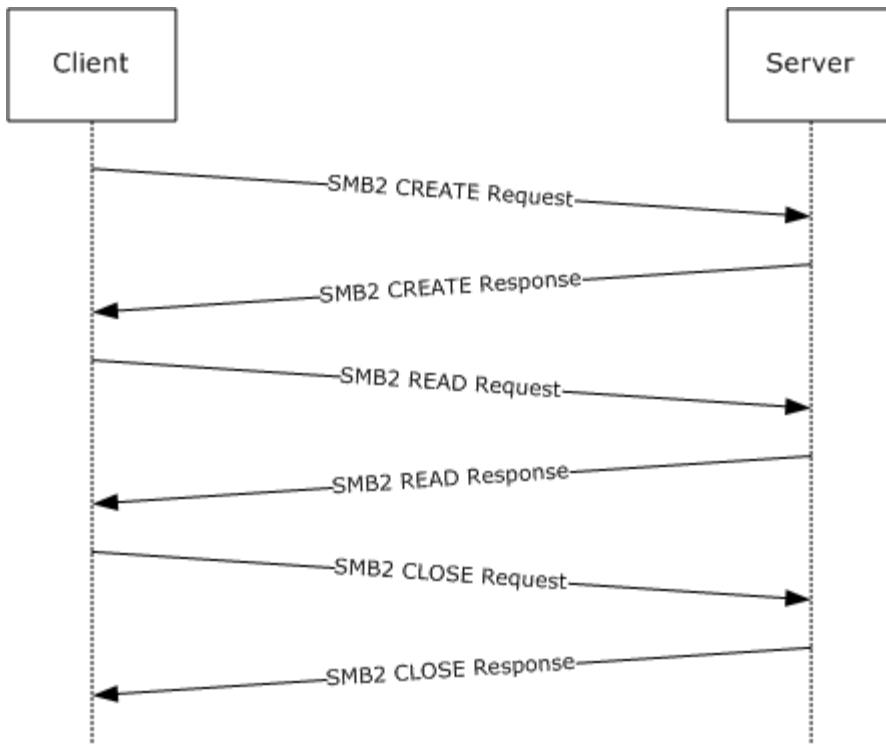


Figure 10: Reading from a remote file

1. The client sends an [SMB2 CREATE Request](#) for the file "testfile.txt".

```

Smb2: C CREATE testfile.txt
SMB2Header:
Size: 64 (0x40)
CreditCharge: 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)
Reserved: 0 (0x0)
TreeId: 5 (0x5)
SessionId: 4398046511121 (0x400000000011)
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)

```

349 / 424

[MS-SMB2] — v20121017
Server Message Block (SMB) Protocol Versions 2 and 3

Copyright © 2012 Microsoft Corporation.

Release: Wednesday, October 17, 2012

```

CreditCharge: 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)
Reserved: 0 (0x0)
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)
CreditCharge: 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)
Reserved: 0 (0x0)
TreeId: 5 (0x5)
SessionId: 4398046511121 (0x4000000001)
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)
CreditCharge: 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)
Reserved: 0 (0x0)
TreeId: 5 (0x5)
SessionId: 4398046511121 (0x4000000001)
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)
CreditCharge: 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)
Reserved: 0 (0x0)
TreeId: 5 (0x5)
SessionId: 4398046511121 (0x400000000011)
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)
CreditCharge: 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)
Reserved: 0 (0x0)
TreeId: 5 (0x5)
```

```

SessionId: 4398046511121 (0x400000000011)
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 0x400000000015 for all requests and responses listed below.

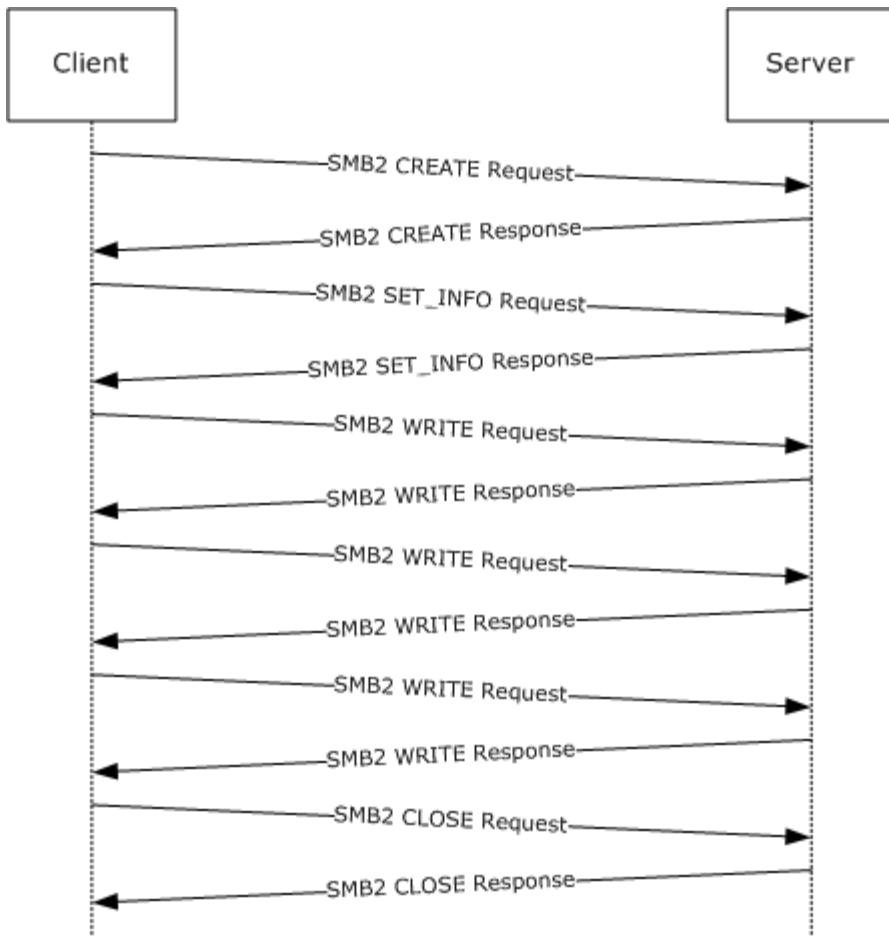


Figure 11: Writing to a remote file

1. The client sends an [SMB2_CREATE Request](#) for the file "test.dat".

```

Smb2: C CREATE test.dat
SMB2Header:
Size: 64 (0x40)
CreditCharge: 0 (0x0)
Status: STATUS_SUCCESS
Command: CREATE
Credits: 111 (0x6F)
Flags: 0 (0x0)
ServerToRedir: .....0 Client to Server
AsyncCommand: .....0. Command is not asynchronous
Related: .....0.. Packet is single message
Signed: .....0... Packet not signed
Reserved: 0 (0x0)
DFS: 0..... Command is not a DFS Operation
NextCommand: 0 (0x0)
MessageId: 10 (0xA)
Reserved: 0 (0x0)
TreeId: 1 (0x1)
SessionId: 4398046511125 (0x40000000015)

```

```

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)
CreditCharge: 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)
Reserved: 0 (0x0)
TreeId: 1 (0x1)
SessionId: 4398046511125 (0x400000000015)
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)
CreditCharge: 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)
Reserved: 0 (0x0)
TreeId: 1 (0x1)
SessionId: 439804651125 (0x40000000015)
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)
CreditCharge: 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)
Reserved: 0 (0x0)
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)
CreditCharge: 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)
Reserved: 0 (0x0)
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)
CreditCharge: 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)
Reserved: 0 (0x0)
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 65536 (0x10000)
SMB2Header:
Size: 64 (0x40)
CreditCharge: 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)
Reserved: 0 (0x0)
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)
CreditCharge: 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)
Reserved: 0 (0x0)
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 131072 (0x20000)
SMB2Header:
Size: 64 (0x40)
CreditCharge: 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)
Reserved: 0 (0x0)
TreeId: 1 (0x1)
```

```
SessionId: 4398046511125 (0x400000000015)
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)
CreditCharge: 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)
Reserved: 0 (0x0)
TreeId: 1 (0x1)
SessionId: 4398046511125 (0x400000000015)
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)
CreditCharge: 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)
Reserved: 0 (0x0)
TreeId: 1 (0x1)
SessionId: 439804651125 (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)
CreditCharge: 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)
Reserved: 0 (0x0)
TreeId: 1 (0x1)
SessionId: 439804651125 (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.

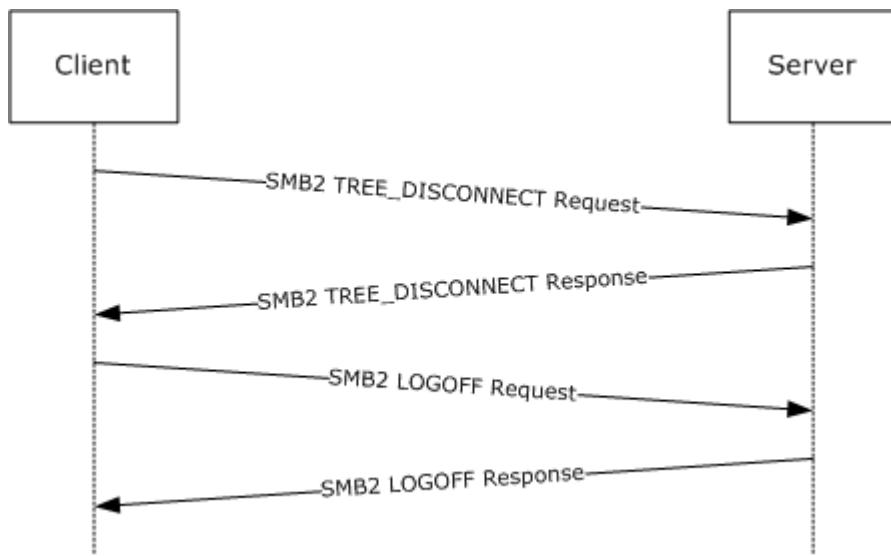


Figure 12: Disconnecting a share and logging off a session

1. The client sends an [SMB2 TREE_DISCONNECT Request](#) for the tree connect.

```

Smb2: C TREE DISCONNECT TID=0x1
SMB2Header:
Size: 64 (0x40)
CreditCharge: 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)
Reserved: 0 (0x0)
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)
CreditCharge: 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)
Reserved: 0 (0x0)
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)
CreditCharge: 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)
Reserved: 0 (0x0)
TreeId: 0 (0x0)
SessionId: 4398046511125 (0x400000000015)
CLogoff:
Size: 4 (0x4)
Reserved: 0 (0x0)
```

4. The server responds with an [SMB2 LOGOFF Response](#) indicating success.

```
Smb2: R LOGOFF
SMB2Header:
Size: 64 (0x40)
CreditCharge: 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)
Reserved: 0 (0x0)
TreeId: 0 (0x0)
SessionId: 4398046511125 (0x400000000015)
RLogoff:
Size: 4 (0x4)
Reserved: 0 (0x0)
```

4.8 Establish Alternate Channel

The following diagram demonstrates the steps taken to establish an alternate **channel**.

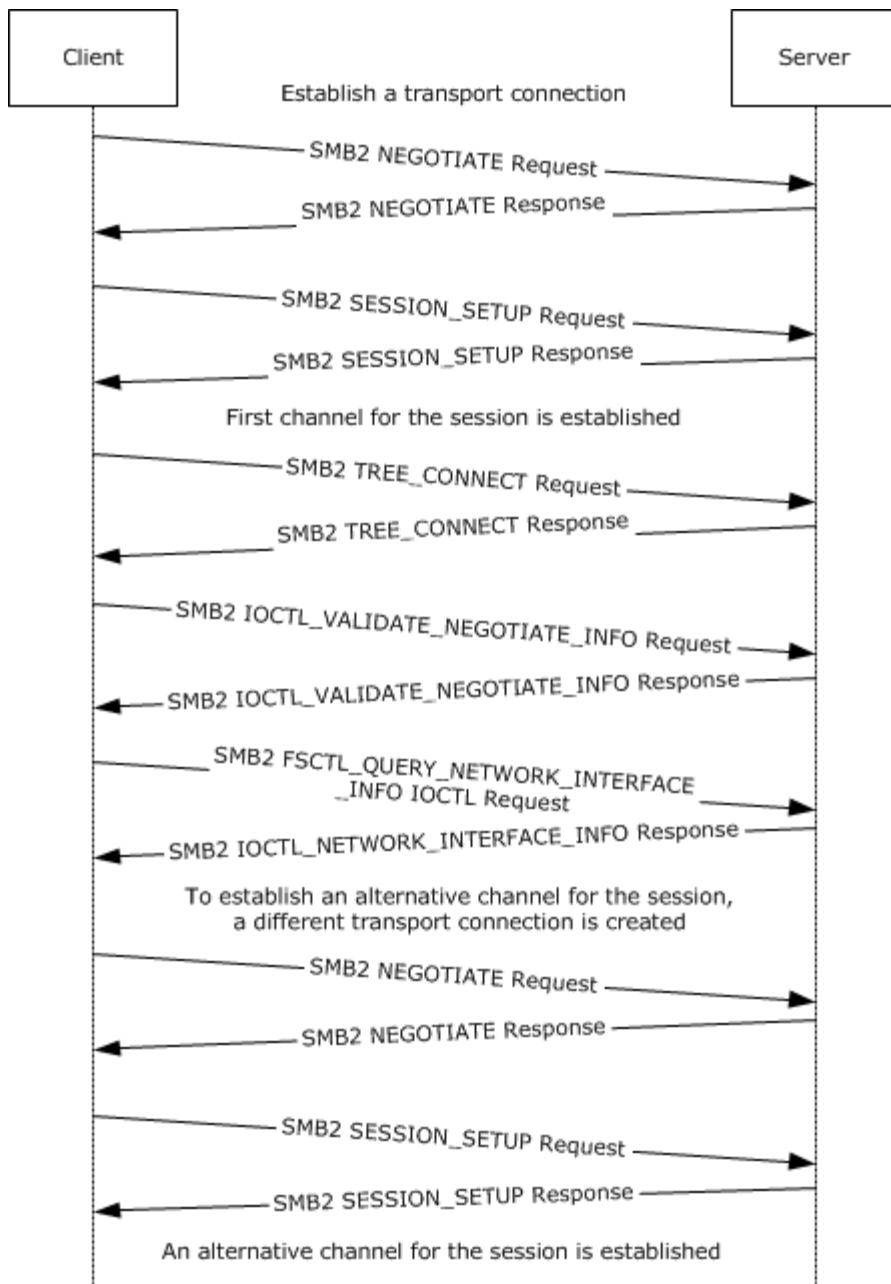


Figure 13: Establishing an alternate channel

1. The client sends an [SMB2_NEGOTIATE Request](#) with dialect 0x300 in the **Dialects** array, and SMB2_GLOBAL_CAP_MULTI_CHANNEL(0x00000008) bit set in **Capabilities**.

```

SMB2: C NEGOTIATE (0x0), ClientGUID={F62E4D0B-C685-E48B-40B6-D815CB56FF6E}
CNegotiate:
StructureSize: 36 (0x24)
DialectCount: 3 (0x3)
SecurityMode: 1 (0x1)
  
```

```

SMB2NEGOTIATESIGNINGENABLED: (.....1) security signatures are enabled on
the client.
SMB2NEGOTIATESIGNINGREQUIRED: (.....0.) security signatures are not required
by the client.
Reserved: (00000000000000..) Reserved
Reserved: 0 (0x0)
Capabilities: 0x7F
ClientGuid: {F62E4D0B-C685-E48B-40B6-D815CB56FF6E}
ClientStartTime: No Time Specified (0)
Dialects:
Dialects: 514 (0x202)
Dialects: 528 (0x210)
Dialects: 768 (0x300)

```

2. The server receives the SMB2 NEGOTIATE Request and finds dialect 0x0300. The server responds with an [SMB2 NEGOTIATE Response](#) with dialect 0x300 in the **DialectRevision**, and the **SMB2_GLOBAL_CAP_MULTI_CHANNEL(0x00000008)** bit set in **Capabilities**.

```

SMB2: R NEGOTIATE (0x0), ServerGUID={1B005379-8063-F0B6-4907-4957998700A1}
SMBidByte: 254 (0xFE)
SMBIdentifier: SMB
SMB2Header: R NEGOTIATE (0x0), TID=0x0000, MID=0x0000, PID=0xFEFF, SID=0x0000
StructureSize: 64 (0x40)
CreditCharge: 0 (0x0)
Status: 0x0, Code = (0) STATUS_SUCCESS, Facility = FACILITY_SYSTEM, Severity =
STATUS_SEVERITY_SUCCESS
Flags: 0x1
NextCommand: 0 (0x0)
MessageId: 0 (0x0)
Reserved: 65279 (0xFEFF)
TreeId: 0 (0x0)
SessionId: 0 (0x0)
Signature: Binary Large Object (16 Bytes)
RNegotiate:
StructureSize: 65 (0x41)
SecurityMode: 1 (0x1)
SMB2NEGOTIATESIGNINGENABLED: (.....1) security signatures are enabled on
the client.
SMB2NEGOTIATESIGNINGREQUIRED: (.....0.) security signatures are not required
by the client.
Reserved: (00000000000000..) Reserved
DialectRevision: (0x300) - SMB 3.0 dialect revision number.
Reserved: 0 (0x0)
ServerGuid: {1B005379-8063-F0B6-4907-4957998700A1}
Capabilities: 0x7F
MaxTransactSize: 1048576 (0x100000)
MaxReadSize: 1048576 (0x100000)
MaxWriteSize: 1048576 (0x100000)
SystemTime: 05/11/2012, 06:41:20.036527 UTC
ServerStartTime: 05/10/2012, 09:56:03.345351 UTC
SecurityBufferOffset: 128 (0x80)
SecurityBufferLength: 120 (0x78)
Reserved2: 0 (0x0)

```

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 (0x1)
CSessionSetup:
StructureSize: 25 (0x19)
Flags: 0 (0x0)
SecurityMode: 1 (0x1)
Capabilities: 0x1
Channel: 0 (0x0)
SecurityBufferOffset: 88 (0x58)
SecurityBufferLength: 74 (0x4A)
PreviousSessionId: 0 (0x0)
securityBlob:
```

4. The server processes the token received with GSS and gets a return code. The GSS return code indicates that an additional exchange is required to complete the authentication. 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 - NT Status: System - Error, Code = (22) STATUS_MORE_PROCESSING_REQUIRED
SESSION SETUP (0x1), SessionFlags=0x0
SMBidByte: 254 (0xFE)
SMBIdentifier: SMB
SMB2Header: R SESSION SETUP (0x1), TID=0x0000, MID=0x0001, PID=0xFFFF, SID=0x4000001
StructureSize: 64 (0x40)
CreditCharge: 0 (0x0)
Status: 0xC0000016, Code = (22) STATUS_MORE_PROCESSING_REQUIRED, Facility =
FACILITY_SYSTEM, Severity = STATUS_SEVERITY_ERROR
Command: SESSION SETUP (0x1)
Credits: 1 (0x1)
Flags: 0x1
NextCommand: 0 (0x0)
MessageId: 1 (0x1)
Reserved: 65279 (0xFFFF)
TreeId: 0 (0x0)
SessionId: 1130302315429889 (0x4040104000001)
Signature: Binary Large Object (16 Bytes)
RSessionSetup:
StructureSize: 9 (0x9)
SessionFlags: 0x0
SecurityBufferOffset: 72 (0x48)
SecurityBufferLength: 349 (0x15D)
```

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 (0x1)
SMBidByte: 254 (0xFE)
SMBIdentifier: SMB
SMB2Header: C SESSION SETUP (0x1), TID=0x0000, MID=0x0002, PID=0xFFFF, SID=0x4000001
StructureSize: 64 (0x40)
CreditCharge: 0 (0x0)
```

```

ChannelSequence: (0x0) - (SMB 3.00 and later only)
Reserved2: 0 (0x0)
Command: SESSION SETUP (0x1)
Credits: 10 (0xA)
Flags: 0x0
NextCommand: 0 (0x0)
MessageId: 2 (0x2)
Reserved: 65279 (0xFEFF)
TreeId: 0 (0x0)
SessionId: 1130302315429889 (0x4040104000001)
Signature: Binary Large Object (16 Bytes)
SMB2: C SESSION SETUP (0x1)
CSessionSetup:
StructureSize: 25 (0x19)
Flags: 0 (0x0)
SecurityMode: 1 (0x1)
Capabilities: 0x1
Channel: 0 (0x0)
SecurityBufferOffset: 88 (0x58)
SecurityBufferLength: 625 (0x271)
PreviousSessionId: 0 (0x0)

```

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 (0x1), SessionFlags=0x0
SMBIdByte: 254 (0xFE)
SMBIdentifier: SMB
SMB2Header: R SESSION SETUP (0x1), TID=0x0000, MID=0x0002, PID=0xFEFF, SID=0x4000001
StructureSize: 64 (0x40)
CreditCharge: 0 (0x0)
Status: 0x0, Code = (0) STATUS_SUCCESS, Facility = FACILITY_SYSTEM, Severity =
STATUS_SEVERITY_SUCCESS
Flags: 0x9
NextCommand: 0 (0x0)
MessageId: 2 (0x2)
Reserved: 65279 (0xFEFF)
TreeId: 0 (0x0)
SessionId: 1130302315429889 (0x4040104000001)
Signature: Binary Large Object (16 Bytes)
RSessionSetup:
StructureSize: 9 (0x9)
SessionFlags: 0x0
SecurityBufferOffset: 72 (0x48)
SecurityBufferLength: 29 (0x1D)

```

7. The client completes the authentication and sends an [SMB2 TREE CONNECT Request](#) with the **SsessionId** for the session, and a tree connect request containing the Unicode share name "\\\smb2server\share".

```

SMB2: C TREE CONNECT (0x3), Path:\\smb2server\\share
SMBIdByte: 254 (0xFE)
SMBIdentifier: SMB

```

```

SMB2Header: C TREE CONNECT (0x3), TID=0x0000, MID=0x0003, PID=0xFEFF, SID=0x4000001
StructureSize: 64 (0x40)
CreditCharge: 0 (0x0)
ChannelSequence: (0x0) - (SMB 3.00 and later only)
Reserved2: 0 (0x0)
Command: TREE CONNECT (0x3)
Credits: 10 (0xA)
Flags: 0x0
NextCommand: 0 (0x0)
MessageId: 3 (0x3)
Reserved: 65279 (0xFEFF)
TreeId: 0 (0x0)
SessionId: 1130302315429889 (0x4040104000001)
Signature: Binary Large Object (16 Bytes)
CTreeConnect:
StructureSize: 9 (0x9)
Reserved: 0 (0x0)
PathOffset: 72 (0x48)
PathLength: 42 (0x2A)
Path:\\smb2server\\share

```

8. The server responds with an [SMB2 TREE CONNECT Response](#) with the **MessageId** of 3, the **CreditResponse** of 5, the **Status** equal to STATUS_SUCCESS, the **SessionId** of 0x8040030000075, and **TreeId** set to the locally generated identifier 0x1.

```

SMB2: R TREE CONNECT (0x3), TID=0x1
SMBIdByte: 254 (0xFE)
SMBIdentifier: SMB
SMB2Header: R TREE CONNECT (0x3), TID=0x0001, MID=0x0003, PID=0xFEFF, SID=0x4000001
StructureSize: 64 (0x40)
CreditCharge: 0 (0x0)
Status: 0x0, Code = (0) STATUS_SUCCESS, Facility = FACILITY_SYSTEM, Severity =
STATUS_SEVERITY_SUCCESS
Flags: 0x1
NextCommand: 0 (0x0)
MessageId: 3 (0x3)
Reserved: 65279 (0xFEFF)
TreeId: 1 (0x1)
SessionId: 1130302315429889 (0x4040104000001)
Signature: Binary Large Object (16 Bytes)
RTreeConnect: 0x1
StructureSize: 16 (0x10)
ShareType: Disk (0x1)
Reserved: 0 (0x0)
ShareFlags: 2048 (0x800)
Capabilities: 0x0
MaximalAccess: 0x1F01FF

```

9. The client sends a FSCTL_VALIDATE_NEGOTIATE_INFO IOCTL request with the **Dialects** array set to 0x202, 0x210, and 0x300, along with the expected server capabilities, security mode, and GUID, to protect against a downgrade attack.

```

SMB2: C IOCTL (0xb), FID=0xFFFFFFFFFFFFFF, FSCTL_VALIDATE_NEGOTIATE_INFO
CIoCtl:

```

```

StructureSize: 57 (0x39)
Reserved: 0 (0x0)
CtlCode: FSCTL_VALIDATE_NEGOTIATE_INFO
FileId: Persistent: 0xFFFFFFFFFFFFFF, Volatile: 0xFFFFFFFFFFFFFF
Persistent: 18446744073709551615 (0xFFFFFFFFFFFFFF)
volatile: 18446744073709551615 (0xFFFFFFFFFFFFFF)
InputOffset: 120 (0x78)
InputCount: 30 (0x1E)
MaxInputResponse: 0 (0x0)
OutputOffset: 120 (0x78)
OutputCount: 0 (0x0)
MaxOutputResponse: 24 (0x18)
Flags: (00000000000000000000000000000001) FSCTL request
Reserved2: 0 (0x0)
ValidateNegotiate:
Capabilities: 0x7F
Guid: {F62E4D0B-C685-E48B-40B6-D815CB56FF6E}
SecurityMode: 1 (0x1)
DialectCount: 3 (0x3)
Dialects:
Dialects: 514 (0x202)
Dialects: 528 (0x210)
Dialects: 768 (0x300)

```

10. The server determines that dialect, capabilities, security mode, and GUID are as expected, and sends an FSCTL_VALIDATE_NEGOTIATE_INFO IOCTL Response with the established values for the connection in an [SMB2 IOCTL Response](#). Upon receiving and validating these, the client successfully validates the end-to-end negotiation and processing proceeds to using the session.

```

SMB2: R IOCTL (0xb), FSCTL_VALIDATE_NEGOTIATE_INFO
SMBidByte: 254 (0xFE)
SMBIdentifier: SMB
SMB2Header: R IOCTL (0xb), TID=0x0001, MID=0x0004, PID=0x000D, SID=0x4000001
StructureSize: 64 (0x40)
CreditCharge: 1 (0x1)
Status: 0x0, Code = (0) STATUS_SUCCESS, Facility = FACILITY_SYSTEM, Severity =
STATUS_SEVERITY_SUCCESS
Flags: 0x9
NextCommand: 0 (0x0)
MessageId: 4 (0x4)
Reserved: 13 (0xD)
TreeId: 1 (0x1)
SessionId: 1130302315429889 (0x4040104000001)
Signature: Binary Large Object (16 Bytes)
RIOCtl:
StructureSize: 49 (0x31)
Reserved: 0 (0x0)
CtlCode: FSCTL_VALIDATE_NEGOTIATE_INFO
FileId: Persistent: 0xFFFFFFFFFFFFFF, Volatile: 0xFFFFFFFFFFFFFF
Persistent: 18446744073709551615 (0xFFFFFFFFFFFFFF)
volatile: 18446744073709551615 (0xFFFFFFFFFFFFFF)
InputOffset: 112 (0x70)
InputCount: 0 (0x0)
OutputOffset: 112 (0x70)
OutputCount: 24 (0x18)
Flags: 0 (0x0)

```

```
Reserved2: 0 (0x0)
ValidateNegotiate:
Capabilities: 0x7F
Dialect: 768 (0x300)
```

11.To establish an alternative channel, the client sends an FSCTL_QUERY_NETWORK_INTERFACE_INFO IOCTL request to query the available network interface on the server.

```
SMB2: C IOCTL (0xb), FID=0xFFFFFFFFFFFFFF, FSCTL_QUERY_NETWORK_INTERFACE_INFO
SMBIdByte: 254 (0xFE)
SMBIdentifier: SMB
SMB2Header: C IOCTL (0xb), TID=0x0001, MID=0x0005, PID=0x000D, SID=0x4000001
StructureSize: 64 (0x40)
CreditCharge: 1 (0x1)
ChannelSequence: (0x0) - (SMB 3.00 and later only)
Reserved2: 0 (0x0)
Command: IOCTL (0xb)
Credits: 10 (0xA)
Flags: 0x0
NextCommand: 0 (0x0)
MessageId: 5 (0x5)
Reserved: 13 (0xD)
TreeId: 1 (0x1)
SessionId: 1130302315429889 (0x4040104000001)
Signature: Binary Large Object (16 Bytes)
CIoCtl:
StructureSize: 57 (0x39)
Reserved: 0 (0x0)
CtlCode: FSCTL_QUERY_NETWORK_INTERFACE_INFO
FileId: Persistent: 0xFFFFFFFFFFFFFF, Volatile: 0xFFFFFFFFFFFFFF
InputOffset: 0 (0x0)
InputCount: 0 (0x0)
MaxInputResponse: 0 (0x0)
OutputOffset: 0 (0x0)
OutputCount: 0 (0x0)
MaxOutputResponse: 1000 (0x3E8)
Flags: (00000000000000000000000000000001) FSCTL request
Reserved2: 0 (0x0)
```

12.The server sends a [NETWORK_INTERFACE_INFO Response](#) in an SMB2 IOCTL Response with the available network interfaces.

```
SMB2: R IOCTL (0xb), FSCTL_QUERY_NETWORK_INTERFACE_INFO
RIoCtl:
StructureSize: 49 (0x31)
Reserved: 0 (0x0)
CtlCode: FSCTL_QUERY_NETWORK_INTERFACE_INFO
FileId: Persistent: 0xFFFFFFFFFFFFFF, Volatile: 0xFFFFFFFFFFFFFF
InputOffset: 112 (0x70)
InputCount: 0 (0x0)
OutputOffset: 112 (0x70)
OutputCount: 912 (0x390)
Flags: 0 (0x0)
Reserved2: 0 (0x0)
InterfaceInfo:
```

```

Next: 152 (0x98)
IfIndex: 12 (0xC)
Capability: 1 (0x1)
RSSCapable: 1 (0x1)
RDMACapable: 0 (0x0)
Reserved: 0 (0x0)
Reserved: 0 (0x0)
LinkSpeed: 10000000000 (0x2540BE400)
SockAddr: 172.25.220.21:0
Family: 2 (0x2)
IPv4: 172.25.220.21:0
Port: 0 (0x0)
Address: 172.25.220.21
Reserved: Binary Large Object (8 Bytes)
EntryPadding: Binary Large Object (112 Bytes)

```

- 13.The client selects any one network interface pair to establish a new connection, and sends an SMB2 NEGOTIATE Request with dialect 0x300 in the **Dialects** array, and SMB2_GLOBAL_CAP_MULTI_CHANNEL(0x00000008) bit set in **Capabilities**.

```

SMB2: C NEGOTIATE (0x0), ClientGUID={F62E4D0B-C685-E48B-40B6-D815CB56FF6E}
SMBidByte: 254 (0xFE)
SMBIdentifier: SMB
SMB2Header: C NEGOTIATE (0x0), TID=0x0000, MID=0x0000, PID=0xFFFF, SID=0x0000
StructureSize: 64 (0x40)
CreditCharge: 0 (0x0)
ChannelSequence: (0x0) - (SMB 3.00 and later only)
Reserved2: 0 (0x0)
Command: NEGOTIATE (0x0)
Credits: 10 (0xA)
Flags: 0x0
NextCommand: 0 (0x0)
MessageId: 0 (0x0)
Reserved: 65279 (0xFFFF)
TreeId: 0 (0x0)
SessionId: 0 (0x0)
Signature: Binary Large Object (16 Bytes)
CNegotiate:
StructureSize: 36 (0x24)
DialectCount: 3 (0x3)
SecurityMode: 1 (0x1)
Reserved: 0 (0x0)
Capabilities: 0x3F
ClientGuid: {F62E4D0B-C685-E48B-40B6-D815CB56FF6E}
ClientStartTime: No Time Specified (0)
Dialects:
Dialects: 514 (0x202)
Dialects: 528 (0x210)
Dialects: 768 (0x300)

```

- 14.The server responds with an SMB2 NEGOTIATE Response with dialect 0x300 in the **DialectRevision**, and SMB2_GLOBAL_CAP_MULTI_CHANNEL(0x00000008) bit set in **Capabilities**.

```
SMB2: R NEGOTIATE (0x0), ServerGUID={1B005379-8063-F0B6-4907-4957998700A1}
```

```

RNegotiate:
StructureSize: 65 (0x41)
SecurityMode: 1 (0x1)
DialectRevision: (0x300) - SMB 3.0 dialect revision number.
Reserved: 0 (0x0)
ServerGuid: {1B005379-8063-F0B6-4907-4957998700A1}
Capabilities: 0x3F
MaxTransactSize: 1048576 (0x100000)
MaxReadSize: 1048576 (0x100000)
MaxWriteSize: 1048576 (0x100000)
SystemTime: 05/11/2012, 06:41:49.996099 UTC
ServerStartTime: 05/10/2012, 09:56:03.345351 UTC
SecurityBufferOffset: 128 (0x80)
SecurityBufferLength: 120 (0x78)
Reserved2: 0 (0x0)

```

- 15.The client sends an SMB2 SESSION_SETUP Request with SMB2_SESSION_FLAG_BINDING set in the **Flags** field and previous channel/session SessionId (0x4040104000001) set in the Header, **PreviousSessionId** field set to 0, and sign the message using Session.SigningKey derived from AES_CMAC-128. Because the request and response are signed, the client does not need to revalidate the negotiation.

```

SMB2: C SESSION SETUP (0x1)
SMBIdByte: 254 (0xFE)
SMBIdentifier: SMB
SMB2Header: C SESSION SETUP (0x1), TID=0x0000, MID=0x0001, PID=0xFFFF, SID=0x4000001
StructureSize: 64 (0x40)
CreditCharge: 0 (0x0)
ChannelSequence: (0x0) - (SMB 3.00 and later only)
Reserved2: 0 (0x0)
Command: SESSION SETUP (0x1)
Credits: 10 (0xA)
Flags: 0x8
NextCommand: 0 (0x0)
MessageId: 1 (0x1)
Reserved: 65279 (0xFFFF)
TreeId: 0 (0x0)
SessionId: 1130302315429889 (0x4040104000001)
Signature: Binary Large Object (16 Bytes)
CSessionSetup:
StructureSize: 25 (0x19)
Flags: 1 (0x1)
SessionBind: (.....1) bind this connection to an existing session (specified in PreviousSessionId)
Reserved: (0000000.) Reserved
SecurityMode: 1 (0x1)
Capabilities: 0x1
Channel: 0 (0x0)
SecurityBufferOffset: 88 (0x58)
SecurityBufferLength: 74 (0x4A)
PreviousSessionId: 0 (0x0)

```

- 16.The server processes the token received with GSS and gets a return code. The GSS return code indicates that an additional exchange is required to complete the authentication. 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 - NT Status: System - Error, Code = (22) STATUS_MORE_PROCESSING_REQUIRED
SESSION SETUP (0x1), SessionFlags=0x0
RSessionSetup:
StructureSize: 9 (0x9)
SessionFlags: 0x0
GU: .....0 NOT a guest user
NU: .....0. NOT a NULL user
Reserved_bits2_15: (00000000000000..) Reserved
SecurityBufferOffset: 72 (0x48)
SecurityBufferLength: 349 (0x15D)
```

- 17.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 response.

```
SMB2: C SESSION SETUP (0x1)
CSessionSetup:
StructureSize: 25 (0x19)
Flags: 1 (0x1)
SessionBind: (.....1) bind this connection to an existing session (specified in PreviousSessionId)
Reserved: (0000000..) Reserved
SecurityMode: 1 (0x1)
Capabilities: 0x1
Channel: 0 (0x0)
SecurityBufferOffset: 88 (0x58)
SecurityBufferLength: 625 (0x271)
PreviousSessionId: 0 (0x0)
```

- 18.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 (0x1), SessionFlags=0x0
SMBIdByte: 254 (0xFE)
RSessionSetup:
StructureSize: 9 (0x9)
SessionFlags: 0x0
GU: .....0 NOT a guest user
NU: .....0. NOT a NULL user
Reserved_bits2_15: (00000000000000..) Reserved
SecurityBufferOffset: 72 (0x48)
SecurityBufferLength: 29 (0x1D)
securityBlob:
```

- 19.An alternate channel has been established for the session.

4.9 Replay Create Request on an Alternate Channel

The following diagram demonstrates the steps taken to replay an [SMB2 CREATE Request](#) on an alternate channel.

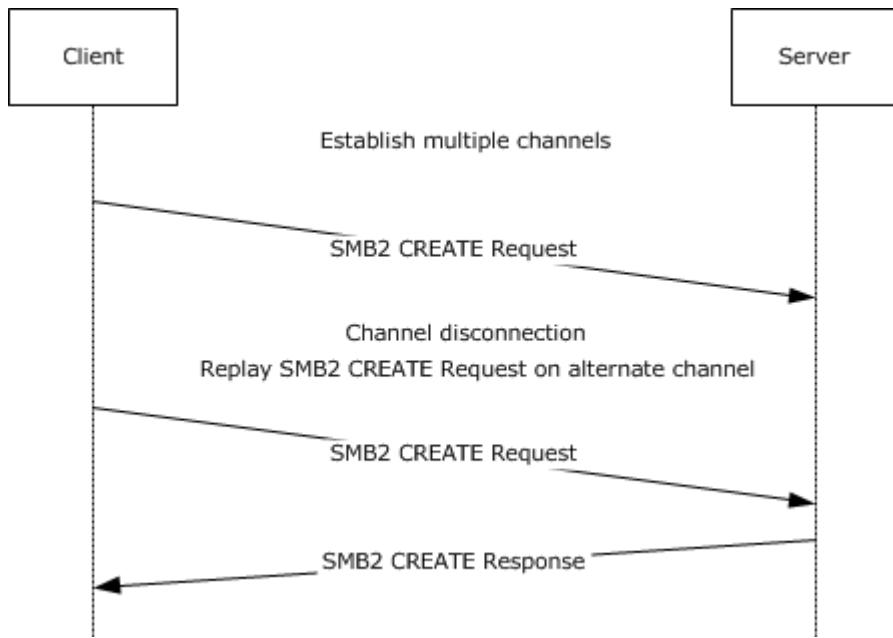


Figure 14: Replay Create Request on an alternate channel

1. The client establishes an alternate channel for a session as described in section [4.8](#)
2. The client sends an SMB2 CREATE Request with
SMB2_CREATE_DURABLE_HANDLE_REQUEST_V2 and SMB2_CREATE_REQUEST_LEASE_V2
create contexts.

```
SMB2: C CREATE (0x5), Da (RW), Sh (RWD), DH2Q+RqLs (RWH-PK), File=Replay.txt@#14
SMBidByte: 254 (0xFE)
SMBIdentifier: SMB
SMB2Header: C CREATE (0x5), TID=0x0001, MID=0x0006, PID=0x000D, SID=0x4000059
StructureSize: 64 (0x40)
CreditCharge: 0 (0x0)
ChannelSequence: (0x0) - (SMB 3.0 and later only)
Reserved2: 0 (0x0)
Command: CREATE (0x5)
Credits: 10 (0xA)
Flags: 0x0
SMB2_FLAGS_REPLAY_OPERATION: (...0.....) Command is a
Replay Operation
NextCommand: 0 (0x0)
MessageId: 6 (0x6)
Reserved: 13 (0xD)
TreeId: 1 (0x1)
SessionId: 1130302315429977 (0x4040104000059)
Signature: Binary Large Object (16 Bytes)
CCreate: 0x1
StructureSize: 57 (0x39)
```

```

SecurityFlags: 0 (0x0)
RequestedOplockLevel: SMB2_OPLOCK_LEVEL_LEASE - A lease is requested.
ImpersonationLevel: Impersonation - The application-requested impersonation level is
Impersonation.
SmbCreateFlags: 0 (0x0)
Reserved: 0 (0x0)
DesiredAccess: 0x12019F
FileAttributes:
FSCCFileAttribute: 32 (0x20)
ShareAccess: Shared for Read/Write/Delete (0x00000007)
CreateDisposition: (0x00000003) Open the file if it already exists; otherwise, create
the file.
CreateOptions: 0x40
NameOffset: 120 (0x78)
NameLength: 20 (0x14)
CreateContextsOffset: 144 (0x90)
CreateContextsLength: 132 (0x84)
Name: Replay.txt
ContextPadding: Binary Large Object (4 Bytes)
Context: DH2Q,Request Durable Handle Open v2
Context:
ECPRequestDurableHandleV2: Request Durable Handle v2
Timeout: 0 (0x0)
Flags: 0 (0x0)
Reserved: (.....0) Reserved
Persistent: (.....0.)
Reserved2: (00000000000000000000000000000000..) Reserved
Reserved: 0 (0x0)
CreateGuid: {33AA3970-EF1A-60A4-4BF1-11F5F9FBBFDB}
Context: RqLs,Lease Request/Response
Context:
CreateRequestLeaseV2: The requested lease state:0x7
LeaseKey: {5A0E33E0-478A-9FA7-4286-B52390B5857B}
LeaseState: 7 (0x7)
READ: (.....1) A read caching lease is requested
HANDLE: (.....1.) A handle caching lease is requested
WRITE: (.....1..) A write caching lease is requested
Reserved: (00000000000000000000000000000000...) Reserved
LeaseFlags: 4 (0x4)
Reserved: (.....00) Reserved
ParentKeyIdValid: (.....1..) Parent lease key field is valid
Reserved2: (00000000000000000000000000000000...) Reserved
LeaseDuration: 0 (0x0)
ParentLeaseKey: {5B4F4EAD-B0E6-B997-4222-50FADEC1FD86}
Epoch: 0 (0x0)

```

3. The connection on which the client sent the SMB2 CREATE request is disconnected; the client cannot receive the [SMB2 CREATE response](#). Since there is another connection on which the same session was bound, the client after a timeout, sends a replay SMB2 CREATE request on that connection. The client sends the SMB2 CREATE request on the alternate channel with the same parameters and create contexts as the original request except that **SMB2_FLAGS_REPLY_OPERATION** bit is set in the **Flags** field of the SMB2 Header.

```

SMB2: C CREATE (0x5), Da (RW), Sh (RWD), DH2Q+RqLs (RWH-PK), File=Replay.txt@#23
SMBIdByte: 254 (0xFE)
SMBIdentifier: SMB

```

```

SMB2Header: C CREATE (0x5), TID=0x0001, MID=0x0006, PID=0x000D, SID=0x4000059
StructureSize: 64 (0x40)
CreditCharge: 0 (0x0)
ChannelSequence: (0x0) - (SMB 3.0 and later only)
Reserved2: 0 (0x0)
Command: CREATE (0x5)
Credits: 10 (0xA)
Flags: 0x0
SMB2_FLAGS_REPLAY_OPERATION: (.1.....) Command is a
Replay Operation
NextCommand: 0 (0x0)
MessageId: 6 (0x6)
Reserved: 13 (0xD)
TreeId: 1 (0x1)
SessionId: 1130302315429977 (0x4040104000059)
Signature: Binary Large Object (16 Bytes)
CCreate: 0x1
StructureSize: 57 (0x39)
SecurityFlags: 0 (0x0)
RequestedOplockLevel: SMB2_OPLOCK_LEVEL_LEASE - A lease is requested.
ImpersonationLevel: Impersonation - The application-requested impersonation level is
Impersonation.
SmbCreateFlags: 0 (0x0)
Reserved: 0 (0x0)
DesiredAccess: 0x12019F
FileAttributes:
FSCCFileAttribute: 32 (0x20)
ShareAccess: Shared for Read/Write/Delete (0x00000007)
CreateDisposition: (0x00000003) Open the file if it already exists; otherwise, create
the file.
CreateOptions: 0x40
NameOffset: 120 (0x78)
NameLength: 20 (0x14)
CreateContextsOffset: 144 (0x90)
CreateContextsLength: 132 (0x84)
Name: Replay.txt
ContextPadding: Binary Large Object (4 Bytes)
Context: DH2Q, Request Durable Handle Open v2
Context:
ECPRequestDurableHandleV2: Request Durable Handle v2
Timeout: 0 (0x0)
Flags: 0 (0x0)
Reserved: (. ....) Reserved
Persistent: (. ....) .
Reserved2: (00000000000000000000000000000000...) Reserved
Reserved: 0 (0x0)
CreateGuid: {33AA3970-EF1A-60A4-4BF1-11F5F9FBBFDB}
Context: RqLs, Lease Request/Response
Context:
CreateRequestLeaseV2: The requested lease state:0x7
LeaseKey: {5AOE33E0-478A-9FA7-4286-B52390B5857B}
LeaseState: 7 (0x7)
READ: (. ....) 1) A read caching lease is requested
HANDLE: (. ....) 1.) A handle caching lease is requested
WRITE: (. ....) 1..) A write caching lease is requested
Reserved: (00000000000000000000000000000000...) Reserved
LeaseFlags: 4 (0x4)
Reserved: (. ....) 00) Reserved
ParentKeyId: (. ....) 1..) Parent lease key field is valid

```

```

Reserved2:          (000000000000000000000000000000...) Reserved
LeaseDuration: 0 (0x0)
ParentLeaseKey: {5B4F4EAD-B0E6-B997-4222-50FADEC1FD86}
Epoch: 0 (0x0)

```

4. The server responds with an SMB2 CREATE response with SMB2_CREATE_DURABLE_HANDLE_REQUEST_V2 and SMB2_CREATE_REQUEST_LEASE_V2 create contexts.

```

SMB2: R  CREATE (0x5), RqLs (RWH-PK)+DH2Q, FID=0x1010000001(Replay.txt@#23)
SMBIdByte: 254 (0xFE)
SMBIdentifier: SMB
SMB2Header: R CREATE (0x5), TID=0x0001, MID=0x0003, PID=0x000D, SID=0x4000059
StructureSize: 64 (0x40)
CreditCharge: 0 (0x0)
Status: 0x0, Code = (0) STATUS_SUCCESS, Facility = FACILITY_SYSTEM, Severity =
STATUS_SEVERITY_SUCCESS
Command: CREATE (0x5)
Credits: 10 (0xA)
Flags: 0x20000001
SMB2_FLAGS_REPLY_OPERATION:           (...1.....) Command is a
Replay Operation
NextCommand: 0 (0x0)
MessageId: 3 (0x3)
Reserved: 13 (0xD)
TreeId: 1 (0x1)
SessionId: 1130302315429977 (0x4040104000059)
Signature: Binary Large Object (16 Bytes)
RCREATE: 0x1
StructureSize: 89 (0x59)
OplockLevel: SMB2_OPLOCK_LEVELLEASE - A lease is requested.
Flags: 0 (0x0)
CreateAction: Opened (0x00000001)
CreationTime: 05/11/2012, 09:23:05.943750 UTC
LastAccessTime: 05/11/2012, 09:23:05.943750 UTC
LastWriteTime: 05/11/2012, 09:23:05.943750 UTC
ChangeTime: 05/11/2012, 09:23:05.943750 UTC
AllocationSize: 0 (0x0)
EndOfFile: 0 (0x0)
FileAttributes:
FSCCFileAttribute: 32 (0x20)
Reserved2: 0 (0x0)
FileId: Persistent: 0x10000010000001D, Volatile: 0x10100000001
Persistent: 72057598332895261 (0x10000010000001D)
volatile: 1103806595073 (0x10100000001)
CreateContextsOffset: 152 (0x98)
CreateContextsLength: 112 (0x70)
Context: RqLs, Lease Request/Response
Context:
CreateResponseLeaseV2: The response lease state:0x087
LeaseKey: {5A0E33E0-478A-9FA7-4286-B52390B5857B}
LeaseState: 7 (0x7)
READ:           (...1.....) A read caching lease is granted
HANDLE:         (...1.....) A handle caching lease is granted
WRITE:          (...1..) A write caching lease is granted
Reserved:      (00000000000000000000000000000000...) Reserved
LeaseFlags: 4 (0x4)

```

```
Reserved1: (.....0) Reserved
BREAK: (.....0.)
ParentKeyValid: (.....1..) Parent lease key field is valid
Reserved: (00000000000000000000000000000000...) Reserved
LeaseDuration: 0 (0x0)
ParentLeaseKey: {5B4F4EAD-B0E6-B997-4222-50FADEC1FD86}
Epoch: 1 (0x1)
ContextPadding: Binary Large Object (4 Bytes)
Context: DH2Q,Request Durable Handle Open v2
Context:
ECPResponseDurableHandleV2: Response Durable Handle V2
Timeout: 60000 (0xEA60)
Flags: 0 (0x0)
Reserved: (.....0) Reserved
Persistent: (.....0.)
Reserved2: (00000000000000000000000000000000..) Reserved
```

5 Security

The following sections specify security considerations for implementers of the SMB 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.

The SMB 2.002 and SMB 2.1 dialects do not support encryption. The SMB 3.0 dialect optionally allows for encryption on a per-share basis. For data that requires stricter security, encryption by the SMB 3.0 protocol is preferred. Alternatively, encryption of the data by the underlying transport must be provided.

5.2 Index of Security Parameters

Security parameter	Section
SHA-256 hashing	3.1.4.1
CMAC-128 hashing	3.1.4.1
Cryptographic key generation	3.1.4.2
SHA-256 hashing	3.1.5.1
CMAC-128 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
- Windows® 8 operating system
- Windows Server® 2012 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 Protocol dialect 2.002 is available only in Windows Vista, Windows Server 2008, Windows 7, Windows Server 2008 R2, Windows 8, and Windows Server 2012. Dialect 2.1 is available only in Windows 7, Windows Server 2008 R2, Windows 8, and Windows Server 2012. Dialect 3.0 is available only in Windows 8 and Windows Server 2012. Windows Vista, Windows Server 2008, Windows 7, Windows Server 2008 R2, Windows 8, and Windows Server 2012 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, Windows Server 2008 R2, Windows 8, and Windows Server 2012 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, Windows Server 2008 R2, Windows 8, and Windows Server 2012 support this dialect revision.

[<9> Section 2.2.3:](#) Only Windows 7, Windows Server 2008 R2, Windows 8, and Windows Server 2012 support this dialect revision.

[<10> Section 2.2.3:](#) Only Windows 8 and Windows Server 2012 support this dialect revision.

[<11> 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.

[<12> Section 2.2.4:](#) Windows Vista, Windows Server 2008, Windows 7, Windows Server 2008 R2, Windows 8, and Windows Server 2012 support this dialect revision.

[<13> Section 2.2.4:](#) Only Windows 7, Windows Server 2008 R2, Windows 8, and Windows Server 2012 support this dialect revision.

[<14> Section 2.2.4:](#) Only Windows 8 and Windows Server 2012 support this dialect revision.

[<15> Section 2.2.4:](#) The "SMB 2.???" dialect string is supported by SMB2 clients and servers in Windows 7 Windows Server 2008 R2, Windows 8, and Windows Server 2012.

[<16> Section 2.2.4:](#) Windows-based SMB2 servers may set this field to any value.

[<17> Section 2.2.4:](#) Windows-based SMB2 servers generate a new **ServerGuid** each time they are started.

[<18> Section 2.2.4:](#) Windows clients do not enforce the **MaxTransactSize** value.

[<19> 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.

[<20> 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.

[<21> 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.

[<22> 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.

[<23> Section 2.2.10:](#) SMB2_SHAREFLAG_FORCE_LEVELII_OPLOCK is not supported on Windows Vista and Windows Server 2008.

[<24> 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.

[<25> Section 2.2.13:](#) When opening a printer file or a named pipe, Windows-based servers ignore these **ShareAccess** values.

[<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:](#) When opening a printer object, Windows-based servers ignore this value.

[<29> 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.

[<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 SMB2 clients do not initialize this bit. The bit contains the value specified by the caller when requesting the open.

[<35> Section 2.2.13:](#) Windows Vista, Windows Server 2008, Windows 7, and Windows 8-based clients will set this bit when it is requested by the application.

[<36> Section 2.2.13.1.1:](#) Windows sets this flag to the value passed in by the higher-level application.

[<37> Section 2.2.13.1.1:](#) Windows 7, Windows Server 2008 R2, Windows 8, and Windows Server 2012 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.

[<38> 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.

[<39> Section 2.2.13.1.2:](#) Windows sets this flag to the value passed in by the higher-level application.

[<40> Section 2.2.13.1.2:](#) Windows 7, Windows Server 2008 R2, Windows 8, and Windows Server 2012 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.

[<41> 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.

[<42> Section 2.2.13.2:](#) If DataLength is 0, Windows-based clients set this field to any value.

[<43> Section 2.2.13.2.8:](#) Windows 7, Windows Server 2008 R2, Windows 8, and Windows Server 2012 as SMB server support the following combinations of values: 0, READ, READ | WRITE, READ |

HANDLE, READ | WRITE | HANDLE. Windows 7, Windows Server 2008 R2, Windows 8, and Windows Server 2012 clients restrict requests accordingly.

[<44> Section 2.2.13.2.10:](#) Windows Server 2012 supports the following combinations of values: 0, READ, READ | WRITE, READ | HANDLE, READ | WRITE | HANDLE. Windows 8 restrict requests accordingly.

[<45> 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.

[<46> Section 2.2.14:](#) Windows-based SMB2 servers always return FILE_OPENED for pipes with successful opens.

[<47> Section 2.2.14:](#) Windows-based SMB2 servers may set this field to any value.

[<48> 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.

[<49> 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.

[<50> 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.

[<51> 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.

[<52> Section 2.2.31:](#) Windows clients set the **OutputOffset** field equal to the **InputOffset** field.

[<53> Section 2.2.31.1.1:](#) Windows sets this field to any value.

[<54> 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.

[<55> Section 2.2.32:](#) Windows-based SMB2 servers set **OutputOffset** to **InputOffset + InputCount**, rounded up to a multiple of 8.

[<56> 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.

[<57> Section 2.2.32.3:](#) Windows-based servers Windows Vista, Windows Server 2008, Windows 7, Windows Server 2008 R2, Windows 8, and Windows Server 2012 always send 4 bytes of zero for the Context field.

[<58> Section 2.2.32.4.1:](#) Windows-based SMB2 servers and clients do not check **SourceFileName**. It is ignored.

[<59> Section 2.2.33:](#) Windows-based servers do not support resuming an enumeration at a specified **FileIndex**. The server will ignore this flag.

[<60> Section 2.2.33:](#) SMB2 wildcard characters are based on Windows wildcard characters, as described in [\[MS-FSA\]](#) section 2.1.4.4, Algorithm for Determining if a FileName Is in an Expression. For more information on wildcard behavior in Windows, see [\[FSBO\]](#) section 7.

[<61> 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.

[<62> 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.

[<63> Section 2.2.37:](#) Windows clients send a 1-byte buffer of 0 when **InputBufferLength** is set to 0.

[<64> Section 2.2.37.1:](#) Windows clients set this field to 1 for TRUE.

[<65> Section 2.2.37.1:](#) Windows clients set this field to 1 for TRUE.

[<66> Section 2.2.37.1:](#) Windows Vista, Windows Server 2008, Windows 7, Windows Server 2008 R2, Windows 8, and Windows Server 2012 clients never send a request using the **SidBuffer** format 2.

[<67> Section 2.2.39:](#) Windows servers will fail the request with STATUS_INVALID_PARAMETER if **BufferOffset** is less than 0x60 or greater than 0xA0.

[<68> Section 3.1.3:](#) By default, Windows-based servers set the [RequireMessageSigning](#) value to TRUE for domain controllers and FALSE for all other machines.

[<69> Section 3.2.1.2:](#) Windows clients do not enforce the **MaxTransactSize** value.

[<70> 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

[<71> 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.

[<72> Section 3.2.3:](#) Windows 8 clients initialize this with "AES-CCM".

[<73> Section 3.2.3:](#) Windows 8 clients set this based on a stored value in the registry.

[<74> 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.

[<75> Section 3.2.4.1.2:](#) Windows Vista SP1, Windows Server 2008, Windows 7, and Windows Server 2008 R2, Windows 8, and Windows Server 2012 clients require a minimum of 4 credits.

[<76> 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.

[<77> Section 3.2.4.1.3:](#) Windows 7, Windows Server 2008 R2, Windows 8, and Windows Server 2012 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.

[<78> Section 3.2.4.1.3:](#) Windows-based clients set the **MessageId** field to 0, when the **AsyncId** field is set to an asynchronous identifier of the request.

[<79> 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.

[<80> Section 3.2.4.1.4:](#) Windows Vista, Windows Server 2008, Windows 7, Windows Server 2008 R2, Windows 8, and Windows Server 2012 will align their compounded requests and responses on 8-byte boundaries. Currently they do not disconnect other machines that disobey this rule.

[<81> Section 3.2.4.1.4:](#) The Windows-based client does not send unrelated compounded requests.

[<82> 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.

[<83> Section 3.2.4.1.5:](#) Windows 7, Windows Server 2008 R2, Windows 8, and Windows Server 2012 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.

[<84> Section 3.2.4.1.5:](#) Windows 7 and Windows 8-based SMB2 clients set the **CreditCharge** field to 1 if **Connection.SupportsMultiCredit** is FALSE.

[<85> Section 3.2.4.1.7:](#) Windows-based clients choose the **Channel** with the least value of **Channel.Connection.OutstandingRequests**.

[<86> Section 3.2.4.1.8:](#) Windows 8 clients choose AES 128 CCM algorithm as specified in [\[RFC4309\]](#).

[<87> Section 3.2.4.2:](#) Windows-based clients always set up a new transport connection when establishing a new session to a server.

[<88> 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.)

[<89> Section 3.2.4.2:](#) Windows will establish a new connection for every SMB2 session being created.

[<90> Section 3.2.4.2:](#) Windows establishes a new connection for each new session.

[<91> Section 3.2.4.2.1:](#) Windows 7, Windows Server 2008 R2, Windows 8, and Windows Server 2012 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.

[<92> 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 implements the SMB 2 Protocol. In the latter case, it will initiate an [SMB2-only negotiate](#).

[<93> 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.

[<94> Section 3.2.4.2.3:](#) Windows-based clients implement the first option that is specified.

[<95> 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*.

[<96> Section 3.2.4.2.3.1:](#) Windows-based clients implement the first option that is specified.

[<97> 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*.

[<98> Section 3.2.4.3:](#) Windows clients set **File.LeaseKey** to a newly generated GUID as specified in [\[MS-DTYP\]](#) section 2.3.2.2.

[<99> Section 3.2.4.3:](#) Windows clients set **File.LeaseKey** to a newly generated GUID as specified in [\[MS-DTYP\]](#) section 2.3.2.2.

[<100> Section 3.2.4.3:](#) Windows-based clients will request a batch oplock for file creates.

[<101> Section 3.2.4.3.5:](#) Windows 8 clients set this to zero.

[<102> 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.

[<103> 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.

[<104> Section 3.2.4.6:](#) Windows Server 2012 defaults to 4 KB

[<105> 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.

[<106> Section 3.2.4.7:](#) Windows Server 2012 defaults to 4 KB

[<107> Section 3.2.4.7:](#) Windows-based clients always put the payload at the beginning of the **Buffer** field and do not insert padding.

[<108> 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.

[<109> Section 3.2.4.9:](#) In a SET_INFO request where **FileInfoClass** is set to FileRenameInformation, Windows clients append an arbitrary number of padding bytes set to arbitrary values at the end of the **Buffer** field.

[<110> 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.

[<111> 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.

[<112> Section 3.2.4.14:](#) Windows Vista, Windows Server 2008, Windows 7, Windows Server 2008 R2, Windows 8, and Windows Server 2012 clients will set **StartSidLength** and **StartSidOffset** to any value.

[<113> 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.

[<114> Section 3.2.4.20.1:](#) Windows clients set this field to any value.

[<115> Section 3.2.4.20.1:](#) Windows clients set the **OutputOffset** field to **InputOffset**.

[<116> Section 3.2.4.20.2.1:](#) Windows clients set this field to any value.

[<117> Section 3.2.4.20.2.1:](#) Windows clients set this field to **InputOffset + InputCount**, rounded up to a multiple of 8 bytes.

[<118> 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.

[<119> Section 3.2.4.20.2.2:](#) Windows clients set the **OutputOffset** field to **InputOffset + InputCount**, rounded up to a multiple of 8 bytes.

[<120> Section 3.2.4.20.3:](#) Windows clients set the **OutputOffset** field to **InputOffset + InputCount**, rounded up to a multiple of 8 bytes.

[<121> Section 3.2.4.20.4:](#) Windows clients set the **OutputOffset** field to **InputOffset + InputCount**, rounded up to a multiple of 8 bytes.

[<122> Section 3.2.4.20.5:](#) Windows clients set this field to any value.

[<123> Section 3.2.4.20.5:](#) Windows clients set the **OutputOffset** field to **InputOffset + InputCount**, rounded up to a multiple of 8 bytes.

[<124> 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.

[<125> Section 3.2.4.20.6:](#) Windows clients set the **OutputOffset** field to **InputOffset + InputCount**, rounded up to a multiple of 8 bytes.

[<126> 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.

[<127> Section 3.2.4.20.8:](#) Windows clients set the **OutputOffset** field to **InputOffset + InputCount**, rounded up to a multiple of 8 bytes.

[<128> Section 3.2.4.20.10:](#) Windows clients set this to 64 kilobytes.

[<129> 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.

[<130> 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.

[<131> Section 3.2.5.1.3:](#) Windows-based clients will not disconnect the connection but simply disregard the incorrectly signed response.

[<132> Section 3.2.5.1.7:](#) Windows-based clients will not disconnect the connection, but will simply fail the request.

[<133> Section 3.2.5.1.8:](#) Windows-based SMB 2 Protocol clients do not check the validity of the command in the response.

[<134> 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.

[<135> Section 3.2.5.5:](#) Windows-based SMB2 clients will try to establish alternate channels, if **Connection.OutstandingRequests** exceeds 8.

[<136> Section 3.2.5.5:](#) Windows-based SMB2 clients will choose the interfaces using the following criteria:

1. For each interface returned in [NETWORK_INTERFACE_INFO Response](#), if the interface has both link-local and non-link-local IP addresses, skip the link-local IP address.
2. If there is one or more multiple link-local addresses (suppose there are Y such interfaces), select local interfaces which only have link-local addresses (suppose there are X such local interfaces).
3. Build a destination address list, include all server non-link-local addresses and X*Y server link-local addresses.
4. For each RDMA capable address pair, duplicate the address pair, one for RDMA and one for Direct TCP.
5. Sort address pairs by which address pair is best suited for connection between client and server.
6. For each address pair, compute
 - Link speed of the pair = min(link speed of local interface, link speed of remote interface)
 - RSS capable = RSS capable of local interface and RSS capable of remote interface
7. If there are RDMA capable address pairs, select them.
 - Otherwise if there are RSS capable address pairs, select them.
 - Otherwise select remaining address pairs.
8. Select the pairs with the highest link speed from the selected address pairs.

9. Select local/remote address pairs so that all eligible local/remote interfaces are used and the connections are distributed among local and remote interfaces.
10. The client attempts to establish an alternate channel on each selected interface and address pair.

[<137> Section 3.2.5.12:](#) Windows 8 and Windows Server 2012 replay the write operation up to three times or until all channels in the session are disconnected.

[<138> Section 3.2.5.14:](#) Windows 8 and Windows Server 2012 replay the IOCTL operation up to three times or until all of the channels in the session are disconnected.

[<139> 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.

[<140> Section 3.2.5.14.9:](#) Windows clients enable TCP keepalives to detect broken connections.

[<141> Section 3.2.5.18:](#) Windows 8 and Windows Server 2012 replay the SetInfo operation up to three times or until all of the channels in the session are disconnected.

[<142> Section 3.2.5.19.1:](#) Windows-based clients will not request exclusive oplocks.

[<143> 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.

[<144> Section 3.2.6.1:](#) Windows clients extend the Request Expiration Timer for asynchronous requests as follows:

If the registry value ExtendedSessTimeout in HKLM\System\CurrentControlSet\Services\LanmanWorkStation\Parameters\ is set, the clients use the same value. Otherwise, the clients extend the expiration time to four times the value of default session timeout.

[<145> Section 3.2.6.1:](#) Windows-based clients return a STATUS_CONNECTION_DISCONNECTED error code to the calling application.

[<146> Section 3.2.6.1:](#) The Windows-based clients will disconnect the connection.

[<147> 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.

[<148> 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.

[<149> Section 3.3.1.2:](#) Windows Vista SP1, Windows Server 2008, Windows 7, Windows Server 2008 R2, Windows 8, and Windows Server 2012 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, Windows 7, and Windows 8	128	2048
Windows Server 2008, Windows Server 2008 R2, and Windows Server 2012	512	8192

[<150> Section 3.3.1.2:](#) A Windows-based server does not currently scale credits based on quality of service features.

[<151> Section 3.3.1.4:](#) Windows 7, Windows Server 2008 R2, Windows 8, and Windows Server 2012-based SMB2 servers support only the levels described above, and Windows 7, Windows Server 2008 R2, Windows 8, and Windows Server 2012-based SMB2 clients request only those levels.

[<152> Section 3.3.1.6:](#) Windows-based servers allow the sharing of both printers and traditional file shares.

[<153> Section 3.3.1.6:](#) In Windows, this abstract state element contains the security descriptor for the share.

[<154> Section 3.3.1.6:](#) Windows-based SMB2 clients do not cache directory enumeration results.

[<155> Section 3.3.1.10:](#) Windows 7, Windows Server 2008 R2, Windows 8, and Windows Server 2012 support a maximum extent of 64 entries in the **Open.LockSequenceArray[]**.

[<156> 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.

[<157> 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).

[<158> Section 3.3.2.2:](#) Windows-based SMB2 servers set this timer to a constant value of 16 minutes.

[<159> Section 3.3.2.3:](#) Windows-based servers implement this timer with a constant value of 45 seconds.

[<160> Section 3.3.3:](#) Windows-based SMB2 servers set this value to 256.

[<161> Section 3.3.3:](#) Windows-based SMB2 servers set this value to 1 MB.

[<162> Section 3.3.3:](#) Windows-based SMB2 servers set this value to 16 MB.

[<163> Section 3.3.3:](#) Windows servers initialize **ServerHashLevel** based on a stored value in the registry.

[<164> Section 3.3.3:](#) Windows 7, Windows Server 2008 R2, Windows 8, and Windows Server 2012 SMB2 servers provide a constant maximum resiliency time-out of 300000 milliseconds.

[<165> Section 3.3.3:](#) Windows 8 and Windows Server 2012 initialize the list with "AES-CCM".

[<166> Section 3.3.3:](#) Windows 8 and Windows Server 2012, by default, set **EncryptData** to FALSE. If the registry value **EncryptData** under HKLM\System\CurrentControlSet\Services\LanmanServer\Parameters\ is set to a nonzero value, **EncryptData** is set to TRUE.

[<167> Section 3.3.3:](#) Windows 8 and Windows Server 2012, by default, set **RejectUnencryptedAccess** to TRUE. If the registry value **RejectUnencryptedAccess** under HKLM\System\CurrentControlSet\Services\LanmanServer\Parameters\ is set to zero, **RejectUnencryptedAccess** is set to FALSE.

[<168> Section 3.3.4.1.1:](#) Windows-based servers do not sign interim responses.

[<169> Section 3.3.4.1.2:](#) For an asynchronously processed request, Windows Vista, Windows Server 2008, Windows 7, Windows Server 2008 R2, Windows 8, and Windows Server 2012 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.

[<170> Section 3.3.4.1.3:](#) The Windows-based server compounds responses for any received compounded operations. Otherwise, it does not compound responses.

[<171> Section 3.3.4.1.3:](#) When there are not enough credits to process a subsequent compounded request, Windows SMB2 servers set the **NextCommand** field to the size of the last SMB2 response message including the [SMB2 header](#).

[<172> 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.

[<173> Section 3.3.4.1.4:](#) Windows Server 2012 choose AES 128 CCM algorithm as specified in [RFC4309].

[<174> 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

[<175> Section 3.3.4.2:](#) Windows-based servers incorrectly process the FSCTL_PIPE_WAIT request on named pipes synchronously.

[<176> Section 3.3.4.2:](#) Windows servers enforce a configurable blocking operation credit, which defaults to 64 on Windows Vista SP1, Windows 7, and Windows 8, and defaults to 512 on Windows Server 2008, Windows Server 2008 R2, and Windows Server 2012.

[<177> 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.

[<178> Section 3.3.4.4:](#) For Windows 7 Windows Server 2008 R2, Windows 8, and Windows Server 2012, 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](#).

[<179> Section 3.3.4.6:](#) Windows-based SMB2 servers set **Open.OplockTimeout** to the current time plus 35000 milliseconds. If **Open.IsPersistent** is TRUE, **Open.OplockTimeout** is set to the current time plus 60000 milliseconds.

[<180> Section 3.3.4.7:](#) Windows based SMB2 servers set **Lease.LeaseBreakTimeout** to the current time plus 35000 milliseconds. If **Open.IsPersistent** is TRUE, **Lease.LeaseBreakTimeout** is set to the current time plus 60000 milliseconds.

[<181> Section 3.3.4.13:](#) Windows Server 2012 sets these bits as appropriate for shared volume configurations.

[<182> Section 3.3.4.13:](#) By default, Windows 8 and Windows Server 2012 set **Share.CATimeout** to zero.

[<183> Section 3.3.4.17:](#) Windows Lease break is described in [\[MS-FSA\]](#) section 2.1.5.17. The *Open* parameter passed is the **Open.Local** value from the current close operation, the *Type* parameter is LEVEL_GRANULAR to indicate a Lease request, and the *RequestedOplockLevel* parameter is zero.

[<184> 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.

[<185> Section 3.3.4.21:](#) On Windows, **ServerName** is used only when the transport is NetBIOS over TCP.

[<186> 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.

[<187> Section 3.3.5.2:](#) Windows performs cancellation of in-progress requests via the interface in [\[MS-FSA\]](#) section 2.1.5.19, Server Requests Canceling an Operation, passing **Request.CancelRequestId** as an input parameter.

[<188> Section 3.3.5.2.1:](#) Windows Server 2012 will accept **OriginalMessageSize** up to 1028 kilobytes.

[<189> Section 3.3.5.2.3:](#) Windows-based servers will not fail the request with an error code but will disconnect clients that send noncompounded requests with an invalid **MessageId**.

[<190> Section 3.3.5.2.4:](#) Windows-based servers will not disconnect the connection due to a mismatched signature.

[<191> Section 3.3.5.2.4:](#) Windows-based servers will not disconnect the connection due to an unsigned packet.

[<192> Section 3.3.5.2.5:](#) Windows Vista, Windows Server 2008, Windows 7, Windows Server 2008 R2, Windows 8, and Windows Server 2012 do not verify the payload size of the request.

[<193> Section 3.3.5.2.6:](#) 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.

[<194> Section 3.3.5.2.7:](#) 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.

[<195> Section 3.3.5.2.7:](#) 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.

[<196> Section 3.3.5.2.7.2:](#) 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.

[<197> Section 3.3.5.2.7.2:](#) Windows Vista, Windows Server 2008, Windows 7, Windows Server 2008 R2, Windows 8, and Windows Server 2012 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, Windows Server 2008 R2, Windows 8, and Windows Server 2012 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.

[<198> Section 3.3.5.2.9:](#) Windows Vista, Windows Server 2008, Windows 7, and Windows Server 2008 R2 servers do not fail the request if the SMB2 header of the request has SMB2_FLAGS_SIGNED set in the **Flags** field and the request is not an SMB2 LOCK request as specified in section [2.2.26](#).

[<199> Section 3.3.5.3.1:](#) If **Connection.SupportsMultiCredit** is TRUE, this is set to a default value of 1024 KB. Otherwise, this is set to a default value of 64 KB.

[<200> 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.

[<201> 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.

[<202> 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.

[<203> Section 3.3.5.3.2:](#) A Windows Vista RTM-based server sets **DialectRevision** to 6.

[<204> Section 3.3.5.3.2:](#) Windows servers set this to a default value of 64KB.

[<205> 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.

[<206> Section 3.3.5.3:](#) Windows 2000, Windows XP, Windows Server 2003, Windows Vista, Windows Server 2008, Windows 7, Windows Server 2008 R2, Windows 8, and Windows Server 2012 will also accept raw Kerberos messages and implicit NTLM messages as part of GSS authentication.

[<207> Section 3.3.5.3:](#) 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.

[<208> Section 3.3.5.7:](#) Windows-based SMB2 servers do not set this bit in the **ShareFlags** field.

[<209> Section 3.3.5.7:](#) Windows-based SMB2 servers do not set this bit in the **ShareFlags** field.

[<210> Section 3.3.5.7:](#) Windows Server 2012 sets these two bits based on group policy settings.

[<211> Section 3.3.5.7:](#) Windows Vista and Windows Server 2008 do not support the SMB2_SHAREFLAG_ENABLE_HASH_V1 bit.

[<212> 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.

[<213> 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.

[<214> Section 3.3.5.9:](#) Windows-based servers fail requests having a **CreateDisposition** of FILE_OPEN or FILE_OVERWRITE, but ignore values of FILE_SUPERSEDE, FILE_OPEN_IF and FILE_OVERWRITE_IF.

[<215> Section 3.3.5.9:](#) Windows Vista, Windows Server 2008, Windows 7, Windows Server 2008 R2, Windows 8, and Windows Server 2012 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](#).

[<216> 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.

[<217> Section 3.3.5.9:](#) Windows Vista and Windows Server 2008 do not support the SMB2_SHAREFLAG_FORCE_LEVELII_OPLOCK flag and ignore the **TreeConnect.Share.ForceLevel2Oplock** value.

[<218> 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 2.1.5.1 Server Requests an Open of a File.

Object Store parameter	SMB2 parameter	Notes
DesiredAccess	DesiredAccess	
DesiredFileAttributes	FileAttributes	

Object Store parameter	SMB2 parameter	Notes
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	A LocalOpen representing TreeConnect.Share.LocalPath . Windows SMB2 servers maintain such a LocalOpen for each active Share.
IsCaseSensitive	FALSE	Windows-based SMB2 servers always handle path names as case-insensitive
TargetOplockKey	NULL	OplockKey specified only for obtaining Leases
ParentOplockKey	NULL	Oplock Key to identify the owner of an oplock on the parent directory of the file being opened.

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

[<219> 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.

[<220> Section 3.3.5.9:](#) Windows **Oplock** acquisition is described in [\[MS-FSA\]](#) section 2.1.5.17. Oplock acquisition is an optional step in open/create processing; the *Open* parameter passed is the **Open.Local** 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
LEVEL_TWO	SMB2_OPLOCK_LEVEL_II

The **Status** code returned indicates whether the requested oplock was granted.

[<221> Section 3.3.5.9:](#) Windows obtains **CreationTime** from the object store FileBasicInformation [MS-FSA] section 2.1.5.11.6 and [MS-FSCC] section 2.4.7.

[<222> Section 3.3.5.9:](#) Windows obtains **LastAccessTime** from the object store FileBasicInformation [MS-FSA] section 2.1.5.11.6 and [MS-FSCC] section 2.4.7.

[<223> Section 3.3.5.9:](#) Windows obtains **LastWriteTime** from the object store FileBasicInformation [MS-FSA] section 2.1.5.11.6 and [MS-FSCC] section 2.4.7.

[<224> Section 3.3.5.9:](#) Windows obtains **ChangeTime** from the object store FileBasicInformation [MS-FSA] section 2.1.5.11.6 and [MS-FSCC] section 2.4.7.

[<225> Section 3.3.5.9:](#) Windows obtains **AllocationSize** from the object store FileStandardInformation [MS-FSA] section 2.1.5.11.27 and [MS-FSCC] section 2.4.38.

[<226> Section 3.3.5.9:](#) Windows-based SMB2 servers will set AllocationSize to any value for the named pipe.

[<227> Section 3.3.5.9:](#) Windows obtains **EndOfFile** from the object store FileStandardInformation [MS-FSA] section 2.1.5.11.27 and [MS-FSCC] section 2.4.38.

[<228> Section 3.3.5.9:](#) Windows-based SMB2 servers will set EndOfFile to any value for the named pipe.

[<229> Section 3.3.5.9:](#) Windows obtains **FileAttributes** from the object store FileBasicInformation [MS-FSA] section 2.1.5.11.6 and [MS-FSCC] section 2.4.7.

[<230> 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.

[<231> 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 2.1.5.9.24 and [MS-FSCC] section 2.3.57.

[<232> 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 2.1.5.16.

[<233> Section 3.3.5.9.2:](#) Windows will ignore security descriptors if the underlying object store does not support them.

[<234> Section 3.3.5.9.3:](#) Windows-based servers support this request.

[<235> Section 3.3.5.9.3:](#) Windows sets allocation size on a newly created file with the FileAllocationInformation [MS-FSA] section 2.1.5.14.1 and [MS-FSCC] section 2.4.4, after converting bytes to volume cluster size.

[<236> 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 2.1.5.5.4.1.

[<237> 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 2.1.5.1.

[<238> Section 3.3.5.9.5:](#) Windows computes the MaximalAccess to return by querying the security attributes of the file with [\[MS-FSA\]](#) section 2.1.5.13, and performing an access check against the credentials provided by the request. **QueryStatus** is set to the **Status** returned in that operation.

[<239> Section 3.3.5.9.6:](#) Windows Vista, Windows 7, Windows Server 2008, and Windows Server 2008 R2 ignore undefined create contexts.

[<240> Section 3.3.5.9.7:](#) Windows Vista, Windows 7, Windows Server 2008, and Windows Server 2008 R2 ignore undefined create contexts.

[<241> Section 3.3.5.9.8:](#) On Windows 7, Windows Server 2008 R2, Windows 8, and Windows Server 2012 ,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. On Windows 8 and Windows Server 2012, the **LeaseKey** in the request is passed to the object store. A new or existing lease is thereby associated with the resulting open.

To acquire or promote the lease as dictated by the SMB2_CREATE_REQUESTLEASE Create Context, a subsequent object store call is invoked as described in [\[MS-FSA\]](#) section 2.1.5.17 Server Requests an Oplock. The Open parameter passed is the **Open.Local** 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	SMB2LEASE_READ_CACHING
WRITE_CACHING	SMB2LEASE_WRITE_CACHING
HANDLE_CACHING	SMB2LEASE_HANDLE_CACHING

The Status code returned indicates whether the requested lease was granted.

[<242> 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.

[<243> Section 3.3.5.9.10:](#) Windows-based servers set the Timeout to 60 seconds.

[<244> Section 3.3.5.9.10:](#) Windows-based SMB2 servers provide a statically configured durable handle time-out, which defaults to 16 minutes.

[<245> Section 3.3.5.9.11:](#) The **LeaseKey** and **ParentLeaseKey** fields in the SMB2_CREATE_REQUESTLEASE_V2 Create Context request are passed to the object store in the form of **TargetOplockKey** and **ParentOplockKey**. A new or existing lease is thereby associated with the resulting open.

To acquire or promote the lease as dictated by the SMB2_CREATE_REQUESTLEASE_V2 Create Context, a subsequent object store call is invoked as described in [\[MS-FSA\]](#) section 2.1.5.17 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** field is constructed to include zero or more bits as follows.

Object Store RequestedOplockLevel bit to be set	SMB2 Lease.LeaseState bit requested
READ_CACHING	SMB2LEASE_READ_CACHING

Object Store RequestedOlockLevel bit to be set	SMB2 Lease.LeaseState bit requested
WRITE_CACHING	SMB2_LEASE_WRITE_CACHING
HANDLE_CACHING	SMB2_LEASE_HANDLE_CACHING

The Status code returned indicates whether the requested lease was granted.

[246> Section 3.3.5.10:](#) Windows obtains attributes and end of file from the object store FileBasicInformation [MS-FSA] section 2.1.5.11.6 and [MS-FSCC] section 2.4.7.

[247> Section 3.3.5.11:](#) Windows flushes any cached data to the file with Server Requests Flushing Cached Data [MS-FSA] section 2.1.5.6.

[248> Section 3.3.5.11:](#) If the request target is a named pipe or file, Windows-based servers handle this request asynchronously.

[249> 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.

[250> Section 3.3.5.12:](#) Windows reads from a file with Server Requests a Read [MS-FSA] section 2.1.5.2.

Object Store parameter	SMB2 parameter
ByteOffset	ByteOffset
ByteCount	ByteCount
Open	Open.Local

[251> 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.

[252> 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.

[253> 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**.

[254> Section 3.3.5.13:](#) Windows Vista, Windows Server 2008, Windows 7, Windows Server 2008 R2, Windows 8, and Windows Server 2012 fail the request with STATUS_BUFFER_OVERFLOW instead of STATUS_INVALID_PARAMETER.

[255> Section 3.3.5.13:](#) Windows Vista, Windows Server 2008, Windows 7, and Windows Server 2008 R2 do not fail WRITE request if **DataOffset** is less than 0x70.

[<256> Section 3.3.5.13:](#) Windows writes to a file with Server Requests a Write [MS-FSA] section 2.1.5.3.

Object Store parameter	SMB2 parameter
ByteOffset	ByteOffset
ByteCount	ByteCount
InputBuffer	Buffer
Open	Open.Local

[<257> 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.

[<258> Section 3.3.5.14:](#) Windows 8 and Windows Server 2012 do not verify the **LockSequence** value in the [SMB2 LOCK Request \(section 2.2.26\)](#) when both **Open.IsResilient** and **Open.IsPersistent** are FALSE.

[<259> Section 3.3.5.14.1:](#) Windows-based servers ignore this value while processing Locks.

[<260> Section 3.3.5.14.1:](#) Windows processes unlock with Server Requests unlock of a Byte-Range [MS-FSA] section 2.1.5.8.

Object Store parameter	SMB2 parameter
FileOffset	Offset
Length	Length
Open	Open.Local

[<261> Section 3.3.5.14.2:](#) Windows-based servers check for SMB2_LOCKFLAG_FAIL_IMMEDIATELY only for the first element of the **Locks** array.

[<262> Section 3.3.5.14.2:](#) Refer to [\[FSBO\]](#) for implementation-specific details of how byte range locks can be implemented.

[<263> Section 3.3.5.14.2:](#) Windows processes lock with Server Requests a Byte-Range Lock [\[MS-FSA\]](#) section 2.1.5.7.

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
Open	Open.Local

[<264> Section 3.3.5.15:](#) Windows Server 2012 allows only the **CtlCode** values as specified in section [2.2.3.1](#) and the following **CtlCode** values, as specified in [\[MS-FSCC\]](#) section 2.3.

FSCTL name	FSCTL function number
FSCTL_CREATE_OR_GET_OBJECT_ID	0x900c0
FSCTL_DELETE_OBJECT_ID	0x900a0
FSCTL_DELETE_REPARSE_POINT	0x900ac
FSCTL_FILESYSTEM_GET_STATISTICS	0x90060
FSCTL_FIND_FILES_BY_SID	0x9008f
FSCTL_GET_COMPRESSION	0x9003c
FSCTL_GET_NTFS_VOLUME_DATA	0x90064
FSCTL_GET_OBJECT_ID	0x9009c
FSCTL_GET_REPARSE_POINT	0x900a8
FSCTL_GET_RETRIEVAL_POINTERS	0x90073
FSCTL_IS_PATHNAME_VALID	0x9002c
FSCTL_LMR_SET_LINK_TRACKING_INFORMATION	0x1400ec
FSCTL_OFFLOAD_READ	0x94264
FSCTL_OFFLOAD_WRITE	0x98268
FSCTL_QUERY_FAT_BPB	0x90058
FSCTL_QUERY_ALLOCATED_RANGES	0x940cf
FSCTL_QUERY_ON_DISK_VOLUME_INFO	0x9013c

FSCTL name	FSCTL function number
FSCTL_QUERY_SPARING_INFO	0x90138
FSCTL_READ_FILE_USN_DATA	0x900eb
FSCTL_SET_COMPRESSION	0x9c040
FSCTL_SET_DEFECT_MANAGEMENT	0x98134
FSCTL_SET_OBJECT_ID	0x90098
FSCTL_SET_OBJECT_ID_EXTENDED	0x900bc
FSCTL_SET_REPARSE_POINT	0x900a4
FSCTL_SET_SPARSE	0x900c4
FSCTL_SET_ZERO_DATA	0x980c8
FSCTL_SET_ZERO_ON_DEALLOCATION	0x90194
FSCTL_WRITE_USN_CLOSE_RECORD	0x900ef

[<265> Section 3.3.5.15:](#) When the connection is on direct TCP transport, Windows 7, Windows Server 2008 R2, Windows 8, and Windows Server 2012 SMB2 servers set this value to 1024*1024 and on NetBIOS over TCP transport, the value is 64*1024.

Windows Vista and Windows Server 2008 SMB2 servers fail an IOCTL request with STATUS_INVALID_PARAMETER if [max(**InputCount**, **MaxInputResponse**) + max(**OutputCount**, **MaxOutputResponse**)] is greater than 256*1024.

[<266> Section 3.3.5.15:](#) Windows 7, Windows Server 2008 R2, Windows 8, and Windows Server 2012 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](#).

[<267> 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.

For the following FSCTLs Windows Vista, Windows Server 2008, Windows 7, and Windows Server 2008 R2 will return STATUS_FILE_CLOSED instead of STATUS_INVALID_DEVICE_REQUEST.

- FSCTL_QUERY_NETWORK_INTERFACE_INFO
- FSCTL_DFS_GET_REFERRALS_EX
- FSCTL_VALIDATE_NEGOTIATE_INFO

[<268> 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 2.1.5.5.4.1.

[<269> 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.

[<270> 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.

[<271> 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.

[<272> 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.

[<273> 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, Windows Server 2008 R2, Windows 8, and Windows Server 2012 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.

[<274> Section 3.3.5.15.3:](#) Windows-based SMB2 servers set **OutputOffset** to **InputOffset + InputCount**, rounded up to a multiple of 8.

[<275> 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.

[<276> Section 3.3.5.15.4:](#) Windows SMB2 servers will set **OutputOffset** to **InputOffset + InputCount**, rounded up to a multiple of 8.

[<277> Section 3.3.5.15.5:](#) Windows servers do not support any additional contexts.

[<278> Section 3.3.5.15.5:](#) Windows servers construct the 24-byte blob using **Open.Durable fileId** and other pieces of information which include the process ID of the caller and a timestamp.

[<279> Section 3.3.5.15.6:](#) Windows performs server-side copy reads from a file with Server Requests a Read [\[MS-FSA\]](#) section 2.1.5.2.

Object Store parameter	SMB2 parameter
ByteOffset	SourceOffset
ByteCount	Length

Object Store parameter	SMB2 parameter
Open	Open.Local

[<280> Section 3.3.5.15.6:](#) Windows servers will return STATUS_INVALID_VIEW_SIZE instead of STATUS_END_OF_FILE.

[<281> Section 3.3.5.15.6:](#) Windows performs server-side copy writes to a file with Application Requests a Write [\[MS-FSA\]](#) section 2.1.5.3.

Object Store parameter	SMB2 parameter
ByteOffset	TargetOffset
ByteCount	Length
InputBuffer	Buffer
Open	Open.Local

[<282> Section 3.3.5.15.7:](#) Windows 7, Windows Server 2008 R2, Windows 8, and Windows Server 2012 servers support the FSCTL_SRV_READ_HASH request.

[<283> 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)

[284](#) [Section 3.3.5.15.8](#): Windows performs passthrough FSCTL operations via Server Requests an FsControl Request [\[MS-FSA\]](#) section 2.1.5.9.

[285](#) [Section 3.3.5.15.8](#): Windows-based SMB2 servers will set **OutputOffset** to **InputOffset** + **InputCount**, rounded up to a multiple of 8.

[286](#) [Section 3.3.5.15.9](#): Windows 7, Windows Server 2008 R2, Windows 8, and Windows Server 2012 servers process the FSCTL_LMR_REQUEST_RESILIENCY request regardless of the negotiated dialect.

[287](#) [Section 3.3.5.15.9](#): Windows 7 and Windows Server 2008 R2 servers keep the resilient handle open indefinitely when the requested **Timeout** value is equal to zero. Windows 8 and Windows Server 2012 servers set a constant value of 120 seconds.

[288](#) [Section 3.3.5.15.13](#): Windows 8 and Windows Server 2012 require that the caller is a member of the Administrators group.

[289](#) [Section 3.3.5.16](#): Windows performs cancellation of in-progress requests via the interface in [\[MS-FSA\]](#) section 2.1.5.19, Server Requests Canceling an Operation, passing **Request.CancelRequestId** as an input parameter.

[290](#) [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.

[291](#) [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.

[292](#) [Section 3.3.5.18](#): Windows-based servers do not support resuming an enumeration at a specified **FileIndex**. The server will ignore this flag.

[293](#) [Section 3.3.5.18](#): Windows performs directory query information requests via the corresponding interfaces in [\[MS-FSA\]](#) section 2.1.5.5.4:

- FileBothDirectoryInformation: [\[MS-FSA\]](#) section 2.1.5.5.4.1.
- FileDirectoryInformation: [\[MS-FSA\]](#) section 2.1.5.5.4.2.
- FileFullDirectoryInformation: [\[MS-FSA\]](#) section 2.1.5.5.4.3.
- FileIdBothDirectoryInformation: [\[MS-FSA\]](#) section 2.1.5.5.4.4.
- FileIdFullDirectoryInformation: [\[MS-FSA\]](#) section 2.1.5.5.4.5.

- FileNamesInformation: [\[MS-FSA\]](#) section 2.1.5.5.4.6.

[**<294> Section 3.3.5.19:**](#) Windows Vista and Windows Server 2008 limit **OutputBufferLength** size to 256 KB.

[**<295> 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.

[**<296> Section 3.3.5.19:**](#) Windows requests ChangeNotify processing via Server Requests Change Notifications for a Directory in [\[MS-FSA\]](#) section 2.1.5.10. 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 2.1.5.10.1.

[**<297> 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.

[**<298> Section 3.3.5.20.1:**](#) Windows-based SMB2 servers will set the **CurrentByteOffset** to any value.

[**<299> 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](#).

[**<300> 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](#).

[**<301> 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.

[**<302> 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 2.1.5.13).

[**<303> Section 3.3.5.20.4:**](#) Windows-based servers do support quotas, if configured.

[**<304> 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 2.1.5.20).

[**<305> 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.

[**<306> Section 3.3.5.21.1:**](#) Windows performs SMB2 SET_INFO SMB2_0_INFO_FILE processing via the subsection of [\[MS-FSA\]](#) section 2.1.5.14 corresponding to the requested FILE_INFORMATION_CLASS value of the **FileInfoClass** request field, as listed in section [2.2.37](#).

[**<307> Section 3.3.5.21.2:**](#) Windows performs SMB2 SET_INFO SMB2_0_INFO_FILESYSTEM processing via the subsection of [\[MS-FSA\]](#) section 2.1.5.15 corresponding to the requested FS_INFORMATION_CLASS value of the **FileInfoClass** request field, as listed in section [2.2.37](#).

[<308> 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.

[<309> 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 2.1.5.16.

[<310> Section 3.3.5.21.4:](#) Windows servers do support quotas, if configured.

[<311> 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 2.1.5.21).

[<312> Section 3.3.7.1:](#) Windows performs cancellation of in-progress requests via the interface in [\[MS-FSA\]](#) section 2.1.5.19, Server Requests Canceling an Operation, passing **Request.CancelRequestId** as an input parameter.

[<313> Section 3.3.7.1:](#) Windows servers set this value to 16 minutes.

[<314> Section 3.3.7.1:](#) Windows performs cancellation of in-progress requests via the interface in [\[MS-FSA\]](#) section 2.1.5.19, 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 July 2012 and October 2012 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 Introduction	Changed cross-reference from [MS-WSO] to [MS-WPO] and changed section cross-references.	N	Content updated.
1.1 Glossary	66448 Added terms "IPv4" and "IPv6" from [MS-GLOS].	Y	Content updated.
1.2.2 Informative References	Added reference [MS-WPO] and reference [MS-WSO].	N	Content updated.
2.2 Message Syntax	Removed underscore character for READ and TRANSFORM_HEADER message categories in the version 3.0 dialect summary.	Y	Content updated.
2.2.4 SMB2 NEGOTIATE Response	66734 Added zero as an acceptable value for the Capabilities field.	Y	Content updated.
2.2.5 SMB2 SESSION SETUP Request	Clarified that the client sets the PreviousSessionId value to the previous session identifier to remove any session	N	Content updated

Section	Tracking number (if applicable) and description	Major change (Y or N)	Change type
	associated with the current identifier.		d.
2.2.13 SMB2 CREATE Request	66761 Removed processing instructions from the FileAttributes field description.	Y	Content updated.
2.2.13 SMB2 CREATE Request	Changed cross-reference from [MS-WSO] to [MS-WPO] and changed section cross-references.	N	Content updated.
2.2.13 SMB2 CREATE Request	66761 Removed statement about field validity in description of FileAttributes field.	Y	Content updated.
2.2.23.2 Lease Break Notification	67056 Clarified the use of the Flags bit values.	Y	Content updated.
2.2.26 SMB2 LOCK Request	66576 Clarified associated handle conflict resolution for byte range locks in SMB2.	Y	Content updated.
2.2.31 SMB2 IOCTL Request	66938 Changed the FileId description to exclude information specific to CtlCode values.	Y	Content updated.
2.2.31.2 SRV_READ_HASH Request	66894 Explained how the meaning of the Length field depends on the hash retrieval type.	N	Content updated.
2.2.32.4.1 HASH_HEADER	66670 Clarified that the assignment of SRV_HASH_VER_2 to the HashVersion field applies only to servers that implement the SMB 3.0 dialect.	Y	Content updated.
2.2.32.4.1 HASH_HEADER	66920 Updated description of the HashBlobLength field.	Y	Content updated.
2.2.32.4.3 SRV_HASH_RETRIEVE_FILE_BASED	66670 Clarified that the SRV_HASH_RETRIEVE_FILE_BASED response is valid for servers that implement the SMB 3.0 dialect.	Y	Content updated.

Section	Tracking number (if applicable) and description	Major change (Y or N)	Change type
<u>2.2.32.5 NETWORK INTERFACE INFO Response</u>	66448 Added cross-reference to new section with socket address information.	Y	Content updated.
<u>2.2.32.5.1 SOCKADDR_STORAGE</u>	66448 Added section.	Y	New content added.
<u>2.2.32.5.1.1 SOCKADDR_IN</u>	66448 Added section.	Y	New content added.
<u>2.2.32.5.1.2 SOCKADDR_IN6</u>	66448 Added section.	Y	New content added.
<u>3.1.4.1 Signing An Outgoing Message</u>	67031 Clarified that the sender copies the 16-byte hash into the SMB2 header.	Y	Content updated.
<u>3.2.4.1.8 Encrypting the Message</u>	66959 Updated "If Session.EncryptData is TRUE ..." processing rule.	Y	Content updated.
<u>3.2.4.2.1 Connecting to the Target Server</u>	Changed cross-reference from [MS-WSO] to [MS-WPO] and changed section cross-references.	N	Content updated.
<u>3.2.4.3 Application Requests Opening a File</u>	66736 Clarified Windows behavior for file entries when the application requests the opening of a file.	Y	New product behavior added.
<u>3.2.4.19 Application Requests Locking of an Array of Byte Ranges</u>	66717 Added a condition governing whether to update the locking requests in the client state.	N	Content updated.
<u>3.3.1.10 Per Open</u>	66738 Added Open.ClientGuid to the list of values to be implemented by SMB 2.1 and 3.0 servers that support leasing.	Y	Content updated.
<u>3.3.4.1</u>	66911 Clarified the server handling of the	Y	Content

Section	Tracking number (if applicable) and description	Major change (Y or N)	Change type
Sending Any Outgoing Message	OutstandingPreRequestCount and OutstandingRequestCount values.		updated.
3.3.4.1.4 Encrypting the Message	66959 Updated "If Session.EncryptData is TRUE ..." processing rule.	Y	Content updated.
3.3.4.1.4 Encrypting the Message	66958 Clarified that the server must encrypt the response if Connection.Dialect is 3.000 and Request.IsEncrypted is TRUE.	Y	Content updated.
3.3.4.7 Object Store Indicates a Lease Break	66984 Revised the case of the Flags field being set to SMB2_NOTIFY_BREAK_LEASE_FLAG_ACK_REQUIRED.	N	Content updated.
3.3.4.13 Server Application Registers a Share	Clarified that the server should set STYPE_CLUSTER_FS, STYPE_CLUSTER_SOFS, and STYPE_CLUSTER_DFS in a manner defined by the implementation and added associated product behavior note.	Y	Content updated.
3.3.5.2.9 Verifying the Session	66675 Changed prescriptive language to state that the server SHOULD fail the request in cases where a session is found and Session.State is Expired. Added product behavior note listing the Windows servers that do not fail the request if additional conditions are met.	Y	Content updated.
3.3.5.2.10 Verifying the Channel Sequence Number	66719 Updated the processing rules for replay operations.	Y	Content updated.
3.3.5.2.10 Verifying the Channel Sequence Number	66919 Clarified processing rules for when there is a 16-bit overflow.	Y	Content updated.
3.3.5.4 Receiving an SMB2 NEGOTIATE Request	66932 Changed server processing instructions to specify failure of the request if the DialectCount field contains 0.	Y	Content updated.
3.3.5.6 Receiving an SMB2 LOGOFF Request	66707 Clarified server behavior for open operations in Session.OpenTable based on the Boolean value of Open.IsDurable.	Y	Content updated.

Section	Tracking number (if applicable) and description	Major change (Y or N)	Change type
3.3.5.7 Receiving an SMB2 TREE CONNECT Request	Changed the name of the SMB2_SHARE_CAP_SCALEOUT bit to SMB2_SHARE_CAP_CLUSTER.	Y	Content updated.
3.3.5.7 Receiving an SMB2 TREE CONNECT Request	66767 Clarified Windows behavior for the conditional setting of the SMB2_SHAREFLAG_ENABLE_HASH_V1 bit.	Y	New product behavior or note added.
3.3.5.9 Receiving an SMB2 CREATE Request	66761 Updated the processing instructions to clarify the server response for pipe opens and print files.	Y	Content updated.
3.3.5.9 Receiving an SMB2 CREATE Request	Updated the processing rules for the FileAttributes field.	Y	Content updated.
3.3.5.9 Receiving an SMB2 CREATE Request	66738 Expanded initialization instructions for the Open object to set Open.ClientGuid to Open.Connection.ClientGuid.	Y	Content updated.
3.3.5.9 Receiving an SMB2 CREATE Request	67082 Clarified the criteria used to validate the file name.	Y	Content updated.
3.3.5.9.7 Handling the SMB2_CREATE_DURABLE_HANDLE_RECONNECT Create Context	66708 Clarified that the server ignores the SMB2_CREATE_DURABLE_HANDLE_REQUEST create context instead of failing it, in the event that the SMB2_CREATE_DURABLE_HANDLE_RECONNECT create context contains one.	Y	Content updated.
3.3.5.9.7 Handling the SMB2_CREATE_DURABLE_HANDLE_RECONNECT Create Context	66707 Clarified server handling of this create context request in cases where Open.Session is not NULL.	Y	Content updated.
3.3.5.9.7 Handling the SMB2_CREATE_DURABLE_HANDLE_RECONNECT Create Context	66738 Changed server processing instructions to specify a failure if Open.ClientGuid is not equal to the ClientGuid of the connection that received the request.	Y	Content updated.
3.3.5.9.8 Handling the SMB2_CREATE_REQUESTLEASE Create	66947 Changed setting of Open.OplockState from	Y	Content updated

Section	Tracking number (if applicable) and description	Major change (Y or N)	Change type
<u>Context</u>	Lease to Held.		d.
<u>3.3.5.9.11 Handling the SMB2_CREATE_REQUESTLEASE V2 Create Context</u>	66947 Changed setting of Open.OplockState from Lease to Held.	Y	Content updated.
<u>3.3.5.9.12 Handling the SMB2_CREATE_DURABLE_HANDLE_RECONNECT V2 Create Context</u>	66707 Amended the processing changes to add that if Open.Session is not NULL, the server must fail the request with STATUS_OBJECT_NAME_NOT_FOUND.	Y	Content updated.
<u>3.3.5.9.12 Handling the SMB2_CREATE_DURABLE_HANDLE_RECONNECT V2 Create Context</u>	66738 Changed server processing instructions to specify a failure if Open.ClientGuid is not equal to the ClientGuid of the connection that received the request.	Y	Content updated.
<u>3.3.5.12 Receiving an SMB2 READ Request</u>	66576 Removed processing rule that a read is to a region that is exclusively locked by another open.	Y	Content updated.
<u>3.3.5.13 Receiving an SMB2 WRITE Request</u>	66576 Removed the processing rule concerning when region is locked by another open.	Y	Content updated.
<u>3.3.5.13 Receiving an SMB2 WRITE Request</u>	66995 Expanded processing rules concerning the server validating the configured maximum write size length and added associated product behavior note.	Y	Content updated.
<u>3.3.5.14 Receiving an SMB2 LOCK Request</u>	66717 Updated the processing of the lock request based on the dialect of SMB2 negotiated with the server, and added a product behavior note concerning the significance of the LockSequence value in the SMB2 LOCK request.	N	Content updated.
<u>3.3.5.15 Receiving an SMB2 IOCTL Request</u>	66938 Changed server processing instructions for CtlCode values of FSCTL DFS_GET_REFERRALS_EX, FSCTL_QUERY_NETWORK_INTERFACE_INFO, and FSCTL_VALIDATE_NEGOTIATE_INFO.	Y	Content updated.
<u>3.3.5.15.1 Handling an Enumeration of Previous Versions Request</u>	67106 Clarified that the server constructs the SRV_SNAPSHOT_ARRAY structure.	Y	Content updated

Section	Tracking number (if applicable) and description	Major change (Y or N)	Change type
			d.
3.3.5.15.7 Handling a Content Information Retrieval Request	66670 Clarified the conditions that result in failure of an SRV_READ_HASH request with an error of STATUS_INVALID_PARAMETER by servers implementing the SMB 2.1 and 3.0 dialects.	Y	Content update d.
3.3.5.15.7 Handling a Content Information Retrieval Request	66920 Added processing rule concerning HashVersion field.	Y	Content update d.
3.3.5.15.7 Handling a Content Information Retrieval Request	66893 Specified when to include a HASH_HEADER in the SRV_READ_HASH response.	N	Content update d.
3.3.5.15.9 Handling a Resiliency Request	67022 Clarified server handling of a resiliency request.	Y	Content update d.
3.3.5.20.3 Handling SMB2_0_INFO_SECURITY	Changed cross-reference from [MS-WSO] to [MS-WPO] and changed section cross-references.	N	Content update d.
3.3.5.20.3 Handling SMB2_0_INFO_SECURITY	Clarified the server response to the client request.	Y	Content update d.
3.3.5.21.3 Handling SMB2_0_INFO_SECURITY	Changed cross-reference from [MS-WSO] to [MS-WPO] and changed section cross-references.	N	Content update d.
3.3.5.22.2 Processing a Lease Acknowledgment	66723 Removed STATUS_FILE_CLOSED from list of return values; added STATUS_OBJECT_NAME_NOT_FOUND.	Y	Content update d.
4.6 Writing to a Remote File	67106 Clarified the client-specified offsets for two SMB2 WRITE Requests.	Y	Content update d.

8 Index

A

Abstract data model
 client ([section 3.1.1](#) 129, [section 3.2.1](#) 131)
 server ([section 3.1.1](#) 129, [section 3.3.1](#) 212)
[Access mask encoding](#) 55
[Applicability](#) 22
[Application Requests Reauthenticating a User](#) 146
[Authenticating the user](#) 145

C

[Capability negotiation](#) 22
[Change notifications algorithm](#) 214
[Change tracking](#) 410
Channel ([section 3.2.1.8](#) 136, [section 3.3.1.14](#) 224)
Client
 abstract data model ([section 3.1.1](#) 129, [section 3.2.1](#) 131)
 [global connections](#) 131
higher-layer triggered events
 [notifying offline status of server](#) 186
 [notifying online status of server](#) 186
 [overview](#) 137
 [re-establishing a durable open](#) 152
 [requesting applying of file attributes](#) 158
 [requesting applying of file security attributes](#) 162
 [requesting applying of file system attributes](#) 160
 [requesting applying of quota information](#) 164
 [requesting cancellation of operation](#) 185
 [requesting change of notifications for directory](#) 167
 [requesting closing of file or named pipe](#) 153
 [requesting closing of share connection](#) 184
 [requesting connection to share](#) 141
 [requesting enumeration of directory](#) 166
 [requesting flushing of cached data](#) 165
 [requesting IO control code operation](#) 170
 [requesting locking of array of byte ranges](#) 168
 [requesting move to server instance](#) 186
 [requesting number of opens on tree connect](#) 185
 [requesting opening of file](#) 148
 [requesting querying for file attributes](#) 157
 [requesting querying for file security attributes](#) 161
 [requesting querying for file system attributes](#) 159
 [requesting querying for quota information](#) 163
 [requesting reading from file or named pipe](#) 154
 [requesting session key for authenticated context](#) 185
 [requesting termination of authenticated context](#) 184
 [requesting unlocking of array of byte ranges](#) 183

[requesting writing to file or named pipe](#) 155
[sending any outgoing message](#) 137
[signing outgoing message](#) 129
initialization ([section 3.1.3](#) 129, [section 3.2.3](#) 136)
local events ([section 3.1.7](#) 131, [section 3.2.7](#) 211)
message processing
 [overview](#) 187
[receiving any message](#) 187
[receiving SMB2 CHANGE NOTIFY response](#) 207
[receiving SMB2 CLOSE response](#) 203
[receiving SMB2 CREATE response for new create operation](#) 199
[receiving SMB2 CREATE response for open reestablishment](#) 202
[receiving SMB2 FLUSH response](#) 203
[receiving SMB2 IOCTL response](#) 204
[receiving SMB2 LOCK response](#) 204
[receiving SMB2 LOGOFF response](#) 196
[receiving SMB2 NEGOTIATE response](#) 189
[receiving SMB2 OPLOCK BREAK notification](#) 208
[receiving SMB2 QUERY DIRECTORY response](#) 207
[receiving SMB2 QUERY INFO response](#) 208
[receiving SMB2 READ response](#) 203
[receiving SMB2 SESSION SETUP response](#) 190
[receiving SMB2 SET INFO response](#) 208
[receiving SMB2 TREE CONNECT response](#) 196
[receiving SMB2 TREE DISCONNECT response](#) 199
[receiving SMB2 WRITE response](#) 204
[verifying incoming message](#) 130
[message sequence numbers algorithm](#) 139
[per channel](#) 136
[per open](#) 134
[per pending request](#) 136
[per session](#) 133
[per SMB2 transport connection](#) 131
[per tree connect](#) 134
[per unique open file](#) 134
[required global data](#) 129
sequencing rules
 [overview](#) 187
 [receiving any message](#) 187
 [receiving SMB2 CHANGE NOTIFY response](#) 207
 [receiving SMB2 CLOSE response](#) 203
 [receiving SMB2 CREATE response for new create operation](#) 199
 [receiving SMB2 CREATE response for open reestablishment](#) 202
 [receiving SMB2 FLUSH response](#) 203
 [receiving SMB2 IOCTL response](#) 204
 [receiving SMB2 LOCK response](#) 204
 [receiving SMB2 LOGOFF response](#) 196
 [receiving SMB2 NEGOTIATE response](#) 189

[receiving SMB2 OPLOCK_BREAK notification](#) 208
[receiving SMB2 QUERY_DIRECTORY response](#) 207
[receiving SMB2_QUERY_INFO response](#) 208
[receiving SMB2_READ response](#) 203
[receiving SMB2_SESSION_SETUP response](#) 190
[receiving SMB2_SET_INFO response](#) 208
[receiving SMB2_TREE_CONNECT response](#) 196
[receiving SMB2_TREE_DISCONNECT response](#) 199
[receiving SMB2_WRITE response](#) 204
[verifying incoming message](#) 130
timer events ([section 3.1.6](#) 131, [section 3.2.6](#) 210)
timers ([section 3.1.2](#) 129, [section 3.2.2](#) 136)
[Connecting to the share](#) 147
[Connecting to the target server](#) 143
[Connections - global](#) 131
[Credit granting algorithm](#) 213

D

[Data - global](#) 129
Data model – abstract
client ([section 3.1.1](#) 129, [section 3.2.1](#) 131)
server ([section 3.1.1](#) 129, [section 3.3.1](#) 212)
[Directory Access Mask packet](#) 57
[Disconnecting example](#) 363
[Durable open scavenger timer](#) 225
[Durable open scavenger timer event](#) 318

E

[Establishing alternate channel example](#) 365
Examples
[disconnecting](#) 363
[establishing alternate channel](#) 365
[logging off](#) 363
[multi-protocol negotiate](#) 321
[named pipe](#) 338
[negotiating SMB 2.10 dialect by using multi-protocol negotiate](#) 327
[overview](#) 321
remote files
[reading](#) 347
[writing](#) 353
[SMB2 negotiate](#) 332

F

[Fields – vendor-extensible](#) 24
[FILE_NOTIFY_INFORMATION packet](#) 117
[File Pipe Printer Access Mask packet](#) 55

G

[Global connections](#) 131
[Global data](#) 129
[Global structures](#) 215
[Glossary](#) 13

H

[HASH_HEADER packet](#) 105
Higher-layer triggered events
client
[notifying offline status of server](#) 186
[notifying online status of server](#) 186
[overview](#) 137
[re-establishing a durable open](#) 152
[requesting applying of file attributes](#) 158
[requesting applying of file security attributes](#) 162
[requesting applying of file system attributes](#) 160
[requesting applying of quota information](#) 164
[requesting cancellation of operation](#) 185
[requesting change of notifications for directory](#) 167
[requesting closing of file or named pipe](#) 153
[requesting closing of share connection](#) 184
[requesting connection to share](#) 141
[requesting enumeration of directory](#) 166
[requesting flushing of cached data](#) 165
[requesting IO control code operation](#) 170
[requesting locking of array of byte ranges](#) 168
[requesting move to server instance](#) 186
[requesting number of opens on tree connect](#) 185
[requesting opening of file](#) 148
[requesting querying for file attributes](#) 157
[requesting querying for file security attributes](#) 161
[requesting querying for file system attributes](#) 159
[requesting querying for quota information](#) 163
[requesting reading from file or named pipe](#) 154
[requesting session key for authenticated context](#) 185
[requesting termination of authenticated context](#) 184
[requesting unlocking of array of byte ranges](#) 183
[requesting writing to file or named pipe](#) 155
[sending any outgoing message](#) 137
[signing outgoing message](#) 129
server
[deregistering share](#) 236
[disabling SMB2 server](#) 240
[enabling SMB2 server](#) 240
[notification that DFS is active](#) 233
[notification that share is DFS share](#) 233
[notification that share is not DFS share](#) 233
[object store indicating lease break](#) 232
[object store indicating oplock break](#) 231
[overview](#) 226
[querying Open](#) 239
[querying session](#) 238
[querying share](#) 236
[querying TreeConnect](#) 238
[registering share](#) 234
[requesting closing of open](#) 237

[requesting closing of session](#) 233
[requesting security context](#) 233
[requesting server statistics](#) 240
[requesting session key](#) 231
[requesting transport binding change](#) 239
[sending any outgoing message](#) 226
[sending error response](#) 230
[sending interim response for asynchronous operation](#) 228
[sending success response](#) 229
[signing outgoing message](#) 129
[updating share](#) 235

I

[Idle connection timer](#) 136
[Idle connection timer event](#) 211
[Implementer - security considerations](#) 381
[Incoming message - verifying](#) 130
[Index of security parameters](#) 381
[Informative references](#) 17
Initialization
 client ([section 3.1.3](#) 129, [section 3.2.3](#) 136)
 server ([section 3.1.3](#) 129, [section 3.3.3](#) 225)
[Introduction](#) 13

L

[Lease](#) 223
[Lease table](#) 223
[Leasing algorithm](#) 214
Local events
 client ([section 3.1.7](#) 131, [section 3.2.7](#) 211)
 server ([section 3.1.7](#) 131, [section 3.3.7](#) 319)
[Logging off example](#) 363

M

Message processing
 client
 [overview](#) 187
 [receiving any message](#) 187
 [receiving SMB2 CHANGE NOTIFY response](#) 207
 [receiving SMB2 CLOSE response](#) 203
 [receiving SMB2 CREATE response for new create operation](#) 199
 [receiving SMB2 CREATE response for open reestablishment](#) 202
 [receiving SMB2 FLUSH response](#) 203
 [receiving SMB2 IOCTL response](#) 204
 [receiving SMB2 LOCK response](#) 204
 [receiving SMB2 LOGOFF response](#) 196
 [receiving SMB2 NEGOTIATE response](#) 189
 [receiving SMB2 OPLOCK BREAK notification](#) 208
 [receiving SMB2 QUERY DIRECTORY response](#) 207
 [receiving SMB2 QUERY INFO response](#) 208
 [receiving SMB2 READ response](#) 203
 [receiving SMB2 SESSION SETUP response](#) 190
 [receiving SMB2 SET INFO response](#) 208
 [receiving SMB2 TREE CONNECT response](#) 196

[receiving SMB2 TREE DISCONNECT response](#) 199
[receiving SMB2 WRITE response](#) 204
[verifying incoming message](#) 130
server
 [accepting incoming connection](#) 241
 [overview](#) 241
 [receiving any message](#) 242
 [receiving SMB COM NEGOTIATE](#) 247
 [receiving SMB2 CANCEL request](#) 302
 [receiving SMB2 CHANGE NOTIFY request](#) 306
 [receiving SMB2 CLOSE request](#) 279
 [receiving SMB2 CREATE request](#) 262
 [receiving SMB2 ECHO request](#) 303
 [receiving SMB2 FLUSH request](#) 280
 [receiving SMB2 IOCTL request](#) 288
 [receiving SMB2 LOCK request](#) 285
 [receiving SMB2 LOGOFF request](#) 258
 [receiving SMB2 NEGOTIATE request](#) 249
 [receiving SMB2 OPLOCK BREAK acknowledgment](#) 316
 [receiving SMB2 QUERY DIRECTORY request](#) 303
 [receiving SMB2 QUERY INFO request](#) 307
 [receiving SMB2 READ request](#) 281
 [receiving SMB2 SESSION SETUP request](#) 251
 [receiving SMB2 SET INFO request](#) 313
 [receiving SMB2 TREE CONNECT request](#) 258
 [receiving SMB2 TREE DISCONNECT request](#) 261
 [receiving SMB2 WRITE request](#) 283
 [verifying incoming message](#) 130
Message sequence numbers algorithm ([section 3.2.4.1.6](#) 139, [section 3.3.1.1](#) 213)
Messages
 [overview](#) 25
 [signing outgoing](#) 129
 [SMB2 Packet Header](#) 26
 [syntax](#) 25
 [transport](#) 25
 [verifying incoming](#) 130
[Multi-protocol negotiate example](#) 321

N

[Named pipe example](#) 338
[Negotiating SMB 2.10 dialect by using multi-protocol negotiate example](#) 327
[Negotiating the protocol](#) 144
[Network disconnect](#) 211
[NETWORK INTERFACE INFO Response packet](#) 108
[NETWORK RESILIENCY REQUEST Request packet](#) 100
[Normative references](#) 16

O

[Open](#) ([section 3.2.1.6](#) 134, [section 3.3.1.10](#) 220)
[Oplock break acknowledgment timer](#) 224
[Oplock break acknowledgment timer event](#) 318
[Outgoing message - signing](#) 129
[Overview \(synopsis\)](#) 18

P

[Parameter index - security](#) 381
[Pending request](#) 136
[Pipe - named - example](#) 338
[Preconditions](#) 21
[Prerequisites](#) 21
[Product behavior](#) 382

R

[References](#)
 [informative](#) 17
 [normative](#) 16
[Relationship to other protocols](#) 20
Remote files
 [reading - example](#) 347
 [writing - example](#) 353
Request 224
[Request expiration timer](#) 136
[Request expiration timer event](#) 210
[Resilient open scavenger timer](#) 225
[Resilient open scavenger timer event](#) 319

S

Security
 [implementer considerations](#) 381
 [overview](#) 381
 [parameter index](#) 381
Sequencing rules
 client
 [overview](#) 187
 [receiving any message](#) 187
 [receiving SMB2 CHANGE NOTIFY response](#) 207
 [receiving SMB2 CLOSE response](#) 203
 [receiving SMB2 CREATE response for new create operation](#) 199
 [receiving SMB2 CREATE response for open reestablishment](#) 202
 [receiving SMB2 FLUSH response](#) 203
 [receiving SMB2 IOCTL response](#) 204
 [receiving SMB2 LOCK response](#) 204
 [receiving SMB2 LOGOFF response](#) 196
 [receiving SMB2 NEGOTIATE response](#) 189
 [receiving SMB2 OPLOCK BREAK notification](#) 208
 [receiving SMB2 QUERY DIRECTORY response](#) 207
 [receiving SMB2 QUERY INFO response](#) 208
 [receiving SMB2 READ response](#) 203
 [receiving SMB2 SESSION SETUP response](#) 190
 [receiving SMB2 SET INFO response](#) 208
 [receiving SMB2 TREE CONNECT response](#) 196
 [receiving SMB2 TREE DISCONNECT response](#) 199
 [receiving SMB2 WRITE response](#) 204
 [verifying incoming message](#) 130
 server
 [accepting incoming connection](#) 241
 [overview](#) 241

[receiving any message](#) 242
[receiving SMB_COM_NEGOTIATE](#) 247
[receiving SMB2 CANCEL request](#) 302
[receiving SMB2 CHANGE NOTIFY request](#) 306
[receiving SMB2 CLOSE request](#) 279
[receiving SMB2 CREATE request](#) 262
[receiving SMB2 ECHO request](#) 303
[receiving SMB2 FLUSH request](#) 280
[receiving SMB2 IOCTL request](#) 288
[receiving SMB2 LOCK request](#) 285
[receiving SMB2 LOGOFF request](#) 258
[receiving SMB2 NEGOTIATE request](#) 249
[receiving SMB2 OPLOCK_BREAK acknowledgment](#) 316
[receiving SMB2 QUERY DIRECTORY request](#) 303
[receiving SMB2 QUERY_INFO request](#) 307
[receiving SMB2 READ request](#) 281
[receiving SMB2 SESSION SETUP request](#) 251
[receiving SMB2_SET_INFO request](#) 313
[receiving SMB2_TREE_CONNECT request](#) 258
[receiving SMB2_TREE_DISCONNECT request](#) 261
[receiving SMB2_WRITE request](#) 283
[verifying incoming message](#) 130
Server
 [abstract data model \(section 3.1.1\)](#) 129, [section 3.3.1](#) 212
 [change notifications algorithm](#) 214
 [credit granting algorithm](#) 213
 [global structures](#) 215
 higher-layer triggered events
 [deregistering share](#) 236
 [disabling SMB2 server](#) 240
 [enabling SMB2 server](#) 240
 [notification that DFS is active](#) 233
 [notification that share is DFS share](#) 233
 [notification that share is not DFS share](#) 233
 [object store indicating lease break](#) 232
 [object store indicating oplock break](#) 231
 [overview](#) 226
 [querying Open](#) 239
 [querying session](#) 238
 [querying share](#) 236
 [querying TreeConnect](#) 238
 [registering share](#) 234
 [requesting closing of open](#) 237
 [requesting closing of session](#) 233
 [requesting security context](#) 233
 [requesting server statistics](#) 240
 [requesting session key](#) 231
 [requesting transport binding change](#) 239
 [sending any outgoing message](#) 226
 [sending error response](#) 230
 [sending interim response for asynchronous operation](#) 228
 [sending success response](#) 229
 [signing outgoing message](#) 129
 [updating share](#) 235
 [initialization \(section 3.1.3\)](#) 129, [section 3.3.3](#) 225)

[leasing algorithm](#) 214
local events ([section 3.1.7](#) 131, [section 3.3.7](#) 319)
message processing
 [accepting incoming connection](#) 241
 [overview](#) 241
 [receiving any message](#) 242
 [receiving SMB_COM_NEGOTIATE](#) 247
 [receiving SMB2 CANCEL request](#) 302
 [receiving SMB2 CHANGE NOTIFY request](#) 306
 [receiving SMB2 CLOSE request](#) 279
 [receiving SMB2 CREATE request](#) 262
 [receiving SMB2 ECHO request](#) 303
 [receiving SMB2 FLUSH request](#) 280
 [receiving SMB2 IOCTL request](#) 288
 [receiving SMB2 LOCK request](#) 285
 [receiving SMB2 LOGOFF request](#) 258
 [receiving SMB2 NEGOTIATE request](#) 249
 [receiving SMB2 OPLOCK_BREAK](#)
 [acknowledgment](#) 316
 [receiving SMB2 QUERY_DIRECTORY request](#) 303
 [receiving SMB2 QUERY_INFO request](#) 307
 [receiving SMB2 READ request](#) 281
 [receiving SMB2 SESSION_SETUP request](#) 251
 [receiving SMB2_SET_INFO request](#) 313
 [receiving SMB2_TREE_CONNECT request](#) 258
 [receiving SMB2_TREE_DISCONNECT request](#) 261
 [receiving SMB2_WRITE request](#) 283
 [verifying incoming message](#) 130
message sequence numbers algorithm 213
per_channel 224
per_lease 223
per_lease_table 223
per_open 220
per_request 224
per_session 219
per_share 217
per_transport_connection 218
per_tree_connect 220
required_global_data 129
sequencing rules
 [accepting incoming connection](#) 241
 [overview](#) 241
 [receiving any message](#) 242
 [receiving SMB_COM_NEGOTIATE](#) 247
 [receiving SMB2 CANCEL request](#) 302
 [receiving SMB2_CHANGE_NOTIFY request](#) 306
 [receiving SMB2_CLOSE request](#) 279
 [receiving SMB2_CREATE request](#) 262
 [receiving SMB2_ECHO request](#) 303
 [receiving SMB2_FLUSH request](#) 280
 [receiving SMB2_IOCTL request](#) 288
 [receiving SMB2_LOCK request](#) 285
 [receiving SMB2_LOGOFF request](#) 258
 [receiving SMB2_NEGOTIATE request](#) 249
 [receiving SMB2_OPLOCK_BREAK](#)
 [acknowledgment](#) 316
 [receiving SMB2_QUERY_DIRECTORY request](#) 303
 [receiving SMB2_QUERY_INFO request](#) 307
 [receiving SMB2_READ request](#) 281
 [receiving SMB2_SESSION_SETUP request](#) 251
 [receiving SMB2_SET_INFO request](#) 313
 [receiving SMB2_TREE_CONNECT request](#) 258
 [receiving SMB2_TREE_DISCONNECT request](#) 261
 [receiving SMB2_WRITE request](#) 283
 [verifying incoming message](#) 130
timer events ([section 3.1.6](#) 131, [section 3.3.6](#) 318)
timers ([section 3.1.2](#) 129, [section 3.3.2](#) 224)
Session ([section 3.2.1.3](#) 133, [section 3.3.1.8](#) 219)
Session_expiration_timer 225
Session_expiration_timer_event 319
Share 217
SMB2_LOCK_Request_packet 92
SMB2_negotiate_example 332
SMB2_Packet_Header 26
SMB2_CANCEL_Request_packet 95
SMB2_CHANGE_NOTIFY_Request_packet 114
SMB2_CHANGE_NOTIFY_Response_packet 116
SMB2_CLOSE_Request_packet 76
SMB2_CLOSE_Response_packet 77
SMB2_CREATE_ALLOCATION_SIZE 72
SMB2_CREATE_ALLOCATION_SIZE_packet 62
SMB2_CREATE_APP_INSTANCE_ID_packet 67
SMB2_CREATE_CONTEXT_Response_Values 71
SMB2_CREATE_CONTEXT_Request_Values_packet 58
SMB2_CREATE_DURABLE_HANDLE_RECONNECT 72
SMB2_CREATE_DURABLE_HANDLE_RECONNECT_packet 62
SMB2_CREATE_DURABLE_HANDLE_RECONNECT_V2_packet 66
SMB2_CREATE_DURABLE_HANDLE_REQUEST_packet 61
SMB2_CREATE_DURABLE_HANDLE_REQUEST_V2_packet 65
SMB2_CREATE_DURABLE_HANDLE_RESPONSE_packet 71
SMB2_CREATE_DURABLE_HANDLE_RESPONSE_V2_packet 76
SMB2_CREATE_FA_BUFFER 71
SMB2_CREATE_QUERY_MAXIMAL_ACCESS_REQUEST_packet 62
SMB2_CREATE_QUERY_MAXIMAL_ACCESS_RESPONSE_packet 72
SMB2_CREATE_QUERY_ON_DISK_ID 64
SMB2_CREATE_QUERY_ON_DISK_ID_packet 73
SMB2_CREATE_Request_packet 50
SMB2_CREATE_REQUEST_LEASE_packet 63
SMB2_CREATE_REQUEST_LEASE_V2_packet 64
SMB2_CREATE_Response_packet 68
SMB2_CREATE_RESPONSE_LEASE_packet 73
SMB2_CREATE_RESPONSE_LEASE_V2_packet 74
SMB2_CREATE_SD_BUFFER 71
SMB2_CREATE_TIMEWARP_TOKEN 72
SMB2_CREATE_TIMEWARP_TOKEN_packet 63
SMB2_ECHO_Request_packet 94
SMB2_ECHO_Response_packet 95

[SMB2_ERROR Response packet](#) 34
[SMB2_FILEID packet](#) 70
[SMB2_FLUSH Request packet](#) 79
[SMB2_FLUSH Response packet](#) 80
[SMB2_IOCTL Request packet](#) 95
[SMB2_IOCTL Response packet](#) 102
[SMB2 Lease Break Acknowledgment packet](#) 89
[SMB2 Lease Break Notification packet](#) 86
[SMB2 Lease Break Response packet](#) 91
[SMB2_LOCK_ELEMENT packet](#) 93
[SMB2_LOCK Request packet](#) 92
[SMB2_LOCK Response packet](#) 94
[SMB2_LOGOFF Request packet](#) 45
[SMB2_LOGOFF Response packet](#) 45
[SMB2_NEGOTIATE Request packet](#) 38
[SMB2_NEGOTIATE Response packet](#) 40
[SMB2_Oplock Break Acknowledgment packet](#) 88
[SMB2_Oplock Break Notification packet](#) 85
[SMB2_Oplock Break Response packet](#) 90
[SMB2_Packet Header ASYNC packet](#) 27
[SMB2_Packet Header SYNC packet](#) 30
[SMB2_QUERY_DIRECTORY Request packet](#) 112
[SMB2_QUERY_DIRECTORY Response packet](#) 114
[SMB2_QUERY_INFO Request packet](#) 118
[SMB2_QUERY_INFO Response packet](#) 123
[SMB2_QUERY_QUOTA_INFO packet](#) 122
[SMB2_READ Request packet](#) 80
[SMB2_READ Response packet](#) 82
[SMB2_SESSION_SETUP Request packet](#) 42
[SMB2_SESSION_SETUP Response packet](#) 44
[SMB2_SET_INFO Request packet](#) 124
[SMB2_SET_INFO Response packet](#) 126
[SMB2_TRANSFORM_HEADER packet](#) 127
[SMB2_TREE_CONNECT Request packet](#) 46
[SMB2_TREE_CONNECT Response packet](#) 46
[SMB2_TREE_DISCONNECT Request packet](#) 49
[SMB2_TREE_DISCONNECT Response packet](#) 50
[SMB2_WRITE Request packet](#) 83
[SMB2_WRITE Response packet](#) 84
[SOCKADDR_IN packet](#) 110
[SOCKADDR_IN6 packet](#) 111
[SOCKADDR_STORAGE packet](#) 109
[SRV_COPYCHUNK packet](#) 99
[SRV_COPYCHUNK_COPY packet](#) 98
[SRV_COPYCHUNK_RESPONSE packet](#) 103
[SRV_HASH_RETRIEVE FILE BASED Response packet](#) 107
[SRV_READ_HASH packet](#) 99
[SRV_READ_HASH response](#) 105
[SRV_READ_HASH Response packet](#) 107
[SRV_REQUEST_RESUME_KEY Response packet](#) 104
[SRV_SNAPSHOT_ARRAY packet](#) 104
[Standards assignments](#) 24
[Symbolic Link Error Response packet](#) 34
[Syntax](#) 25

T

[Timer events](#)
client ([section 3.1.6](#) 131, [section 3.2.6](#) 210)
server ([section 3.1.6](#) 131, [section 3.3.6](#) 318)

[Timers](#)

client ([section 3.1.2](#) 129, [section 3.2.2](#) 136)
server ([section 3.1.2](#) 129, [section 3.3.2](#) 224)
[Tracking changes](#) 410
[Transport](#)
[connection](#) 218
[disconnect](#) 319
[messages](#) 25
[Transport connection](#) 131
Tree connect ([section 3.2.1.4](#) 134, [section 3.3.1.9](#) 220)
Triggered events – higher layer
client
[notifying offline status of server](#) 186
[notifying online status of server](#) 186
[overview](#) 137
[re-establishing a durable open](#) 152
[requesting applying of file attributes](#) 158
[requesting applying of file security attributes](#) 162
[requesting applying of file system attributes](#) 160
[requesting applying of quota information](#) 164
[requesting cancellation of operation](#) 185
[requesting change of notifications for directory](#) 167
[requesting closing of file or named pipe](#) 153
[requesting closing of share connection](#) 184
[requesting connection to share](#) 141
[requesting enumeration of directory](#) 166
[requesting flushing of cached data](#) 165
[requesting IO control code operation](#) 170
[requesting locking of array of byte ranges](#) 168
[requesting move to server instance](#) 186
[requesting number of opens on tree connect](#) 185
[requesting opening of file](#) 148
[requesting querying for file attributes](#) 157
[requesting querying for file security attributes](#) 161
[requesting querying for file system attributes](#) 159
[requesting querying for quota information](#) 163
[requesting reading from file or named pipe](#) 154
[requesting session key for authenticated context](#) 185
[requesting termination of authenticated context](#) 184
[requesting unlocking of array of byte ranges](#) 183
[requesting writing to file or named pipe](#) 155
[sending any outgoing message](#) 137
[signing outgoing message](#) 129
server
[deregistering share](#) 236
[disabling SMB2 server](#) 240
[enabling SMB2 server](#) 240
[notification that DFS is active](#) 233
[notification that share is DFS share](#) 233
[notification that share is not DFS share](#) 233
[object store indicating lease break](#) 232
[object store indicating oplock break](#) 231

[overview](#) 226
[querying Open](#) 239
[querying session](#) 238
[querying share](#) 236
[querying TreeConnect](#) 238
[registering share](#) 234
[requesting closing of open](#) 237
[requesting closing of session](#) 233
[requesting security context](#) 233
[requesting server statistics](#) 240
[requesting session key](#) 231
[requesting transport binding change](#) 239
[sending any outgoing message](#) 226
[sending error response](#) 230
[sending interim response for asynchronous operation](#) 228
[sending success response](#) 229
[signing outgoing message](#) 129
[updating share](#) 235

U

[Unique open file](#) 134

V

[VALIDATE_NEGOTIATE_INFO_Request_packet](#) 101
[VALIDATE_NEGOTIATE_INFO_Response_packet](#) 111
[Vendor-extensible fields](#) 24
[Versioning](#) 22