[MS-FSA]:
File System Algorithms

Intellectual Property Rights Notice for Open Specifications Documentation

Technical Documentation. Microsoft publishes Open Specifications documentation for protocols, file formats, languages, standards as well as overviews of the interaction among each of these technologies.

Copyrights. This documentation is covered by Microsoft copyrights. Regardless of any other terms that are contained in the terms of use for the Microsoft website that hosts this documentation, you may make copies of it in order to develop implementations of the technologies described in the Open Specifications and may distribute portions of it in your implementations using these technologies or your documentation as necessary to properly document the implementation. You may also distribute in your implementation, with or without modification, any schema, IDL's, or code samples that are included in the documentation. This permission also applies to any documents that are referenced in the Open Specifications.

No Trade Secrets. Microsoft does not claim any trade secret rights in this documentation.

Patents. Microsoft has patents that may cover your implementations of the technologies described in the Open Specifications. Neither this notice nor Microsoft's delivery of the documentation grants any licenses under those or any other Microsoft patents. However, a given Open Specification may be covered by Microsoft Open Specification Promise or the Community Promise. If you would prefer a written license, or if the technologies described in the Open Specifications are not covered by the Open Specifications Promise or Community Promise, as applicable, patent licenses are available by contacting iplg@microsoft.com.

Trademarks. The names of companies and products contained in this documentation may be covered by trademarks or similar intellectual property rights. This notice does not grant any licenses under those rights.

Fictitious Names. The example companies, organizations, products, domain names, e-mail addresses, logos, people, places, and events depicted in this documentation are fictitious. No association with any real company, organization, product, domain name, email address, logo, person, place, or event is intended or should be inferred.

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

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

Preliminary Documentation. This Open Specification provides documentation for past and current releases and/or for the pre-release (beta) version of this technology. This Open Specification is final

documentation for past or current releases as specifically noted in the document, as applicable; it is preliminary documentation for the pre-release (beta) versions. Microsoft will release final documentation in connection with the commercial release of the updated or new version of this technology. As the documentation may change between this preliminary version and the final version of this technology, there are risks in relying on preliminary documentation. To the extent that you incur additional development obligations or any other costs as a result of relying on this preliminary documentation, you do so at your own risk.

Revision Summary

Date	Revision History	Revision Class	Comments
03/12/2010	0.1	Major	First Release.
04/23/2010	0.1.1	Editorial	Revised and edited the technical content.
06/04/2010	1.0	Major	Updated and revised the technical content.
07/16/2010	2.0	Major	Significantly changed the technical content.
08/27/2010	3.0	Major	Significantly changed the technical content.
10/08/2010	4.0	Major	Significantly changed the technical content.
11/19/2010	5.0	Major	Significantly changed the technical content.
01/07/2011	6.0	Major	Significantly changed the technical content.
02/11/2011	6.0	No change	No changes to the meaning, language, or formatting of the technical content.
03/25/2011	6.0	No change	No changes to the meaning, language, or formatting of the technical content.
05/06/2011	7.0	Major	Significantly changed the technical content.
06/17/2011	8.0	Major	Significantly changed the technical content.
09/23/2011	9.0	Major	Significantly changed the technical content.
12/16/2011	10.0	Major	Significantly changed the technical content.
03/30/2012	11.0	Major	Significantly changed the technical content.

Contents

1 Introduction

This document defines an abstract model for how an object store can be implemented to support the Common Internet File System (CIFS) Protocol, the Server Message Block (SMB) Protocol, and the Server Message Block (SMB) Version 2 Protocol (described in [MS-CIFS], [MS-SMB] and [MS-SMB2], respectively).

Section 2 of this specification is normative and can contain the terms MAY, SHOULD, MUST, MUST NOT, and SHOULD NOT as defined in RFC 2119. Section 1.6 is 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-FSCC]:

cluster

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

volume
globally unique identifier (GUID)
mount point
reparse point
server
SID
symbolic link
Unicode

The following terms are specific to this document:

Alternate Data Stream: A named data stream that is part of a file or directory, which can be opened independently of the **default data stream**. Many operations on an alternate data stream affect only that stream and not other streams or the file or directory as a whole.

Backup: The act of copying data (usually files) to some other storage media in case of equipment failure or other catastrophic event.

Compression Unit: A segment of a stream that the object store can compress, encrypt, or make sparse independently of other segments of the same stream.

Default Data Stream: The unnamed data stream in a non-directory file. Many operations on a default data stream affect the file as a whole.

Restore: The act of copying data (usually files) back to its original storage location from some other storage media after some form of data loss.

Software Defect Management: A mechanism for the object store to manage and remap defective blocks on removable rewritable media (such as CD-RW, DVD-RW, and DVD+RW).<1>

WinPE: Windows Pre-installation Environment.

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

1.2 References

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

1.2.1 Normative References

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

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

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

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

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

[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

[RFC4122] Leach, P., Mealling, M., and Salz, R., "A Universally Unique Identifier (UUID) URN Namespace", RFC 4122, July 2005, http://www.ietf.org/rfc/rfc4122.txt

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

[INCITS-T10/11-059] INCITS, "T10 specification 11-059", http://www.t10.org/cgi-bin/ac.pl?t=d&f=11-059r9.pdf

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

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

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

[MS-SMB2] Microsoft Corporation, "Server Message Block (SMB) Protocol Versions 2 and 3 Specification".

[SIS] Microsoft Corporation, "Single Instance Storage in Microsoft Windows Storage Server 2003 R2", May 2006, http://www.microsoft.com/technet/itshowcase/content/sistwp.mspx

1.3 Overview

None.

1.4 Relationship to Other Protocols

None.

1.5 Prerequisites/Preconditions

None.

1.6 Applicability Statement

None.

1.7 Versioning and Capability Negotiation

None.

1.8 Vendor-Extensible Fields

This algorithm uses NTSTATUS values as defined in [MS-ERREF] section 2.3. Vendors are free to choose their own values for this field, as long as the C bit (0x20000000) is set, indicating it is a customer code.

1.9 Standards Assignments

2 Messages

This is an algorithms document describing wire-visible behavior of a backing object store that is referenced by the following protocol documents:

The Common Internet File System (CIFS) Protocol Specification [MS-CIFS]

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

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

3 Algorithm Details

3.1 Object Store 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 algorithm. The described organization is provided to facilitate the explanation of how the algorithm behaves. This document does not mandate that implementations adhere to this model as long as their external behavior is consistent with that described in this document.

The following abstract object types are defined in this document:

Volume

TunnelCacheEntry

File

Link

Stream

Open

ByteRangeLock

ChangeNotifyEntry

NotifyEventEntry

Oplock

RHOpContext

CancelableOperations

SecurityContext

The following shorthand forms are also used:

DataFile: A **File** object with a FileType of DataFile.

DirectoryFile: A **File** object with a FileType of DirectoryFile.

DataStream: A **Stream** object with a StreamType of DataStream.

DirectoryStream: A **Stream** object with a StreamType of DirectoryStream.

Plural forms of all these object types are also used.

3.1.1.1 Per Volume

Note: Some of the information in this section is subject to change because it applies to a preliminary implementation of the protocol or structure. For information about specific differences between versions, see the behavior notes that are provided in the Product Behavior appendix.

The object store MUST implement the following persistent attributes:

RootDirectory: The **DirectoryFile** for the root of this **volume**.

TotalSpace: A 64-bit unsigned integer specifying the total size of the volume in bytes. This value MUST be a multiple of **ClusterSize**.

FreeSpace: A 64-bit unsigned integer specifying the available space of the volume in bytes. This value MUST be a multiple of **ClusterSize**.

IsReadOnly: A Boolean that is TRUE if the volume is read-only and MUST NOT be modified; otherwise, the volume is both readable and writable.

IsQuotasSupported: A Boolean that is TRUE if the physical media format for this volume supports Quotas.

IsObjectIDsSupported: A Boolean that is TRUE if the physical media format for this volume supports ObjectIDs.

IsReparsePointsSupported: A Boolean that is TRUE if the physical media format for this volume supports ReparsePoints.

VolumeLabel: A 16-character **Unicode** string containing the name of the volume. An empty value is supported.

LogicalBytesPerSector: A 32-bit unsigned integer specifying the size of a sector for this volume in bytes. **LogicalBytesPerSector** MUST be a power of two and MUST be greater than or equal to 512 and less than or equal to **Volume.SystemPageSize**.

ClusterSize: A 32-bit unsigned integer specifying the size of a **cluster** for this volume in bytes. **ClusterSize** MUST be a power of two, and MUST be greater than or equal to **LogicalBytesPerSector** and a power-of-two multiple of **LogicalBytesPerSector**.<2>

PhysicalBytesPerSector: A 32-bit unsigned integer specifying the size of a physical sector for this volume in bytes. **PhysicalBytesPerSector** MUST be a power of two, MUST be greater than or equal to 512 and less than or equal to **Volume.SystemPageSize**, and MUST be greater than or equal to **Volume.LogicalBytesPerSector**.

PartitionOffset: A 64-bit unsigned integer specifying the byte offset from the first physical sector where the first partition is placed.

SystemPageSize: A 32-bit unsigned integer specifying the size, in bytes, of a page of memory in the system. This value is architecture dependent.<3>

VolumeCreationTime: The time the volume was formatted in the FILETIME format specified in [MS-FSCC] section 2.1.1.

VolumeSerialNumber: A 32-bit unsigned integer that contains a number, randomly generated at format time, to uniquely identify the volume.

VolumeCharacteristics: A bit field identifying various characteristics about the current volume as specified in [MS-FSCC] section 2.5.10.

CompressionUnitSize: A 32-bit unsigned integer specifying the **compression unit** size in bytes, which is the granularity used when compressing, encrypting, or sparsifying portions of a stream independent of other portions of the same stream. Not all file systems support these features, and implementation of this field is optional. If one or more of these features are

supported, the value of this field is implementation-defined but MUST be a power of two multiple of **ClusterSize**.<4>

CompressedChunkSize: A 32-bit unsigned integer specifying the maximum size of each chunk in a compressed stream. Not all file systems support compression, and implementation of this field is optional. If compression is supported, the value of this field is implementation-defined but MUST be a power of two and MUST be less than or equal to **CompressionUnitSize**.<5>

ChecksumChunkSize: A 32-bit unsigned integer that specifies the size of each chunk in a stream that is configured with integrity. Not all file systems support integrity, and implementation of this field is optional.<6>

TunnelCacheList: A list of zero or more **TunnelCacheEntries** providing metadata about recently deleted or renamed files. The list could be empty if the object store does not implement tunnel caching or if there are no recently deleted or renamed files on this volume.

ChangeNotifyList: A list of zero or more **ChangeNotifyEntries** describing outstanding change notify requests for the volume.

GenerateShortNames: A Boolean that is TRUE if short name creation support is enabled on this Volume. FALSE if short name creation is not supported on this Volume.

QuotaInformation: A list of FILE_QUOTA_INFORMATION elements (per [MS-FSCC] section 2.4.33) that track the total **Stream.AllocationSize** per SID where the **File.SecurityDescriptor.Owner** field is equal to the SID.<7>

DefaultQuotaThreshold: A 64-bit signed integer that contains the default per-user disk quota warning threshold in bytes. Not all file systems support this field, and implementation of this field is optional.

DefaultQuotaLimit: A 64-bit signed integer that contains the default per-user disk quota limit in bytes. Not all file systems support this field, and implementation of this field is optional.

VolumeQuotaState: A bitmask of flags defining the current quota state on the volume as specified in [MS-FSCC] section 2.5.2 under FileSystemControlFlags. Not all file systems support this field, and implementation of this field is optional.

VolumeId: A **GUID** as specified in [RFC4122]. This value MAY be NULL.

ExtendedInfo: A 48-byte structure containing extended VolumeId information, as described in [MS-FSCC] section 2.5.6.<8>

IsUsnJournalActive: A Boolean that is TRUE if a USN change journal is active on the volume.<9>

LastUsn: A 64-bit unsigned integer indicating the positive USN number of the last record written to the USN change journal on the volume, or 0 if no USN records have been written. If **IsUsnJournalActive** is FALSE, **LastUsn** MUST be 0.

IsOffloadReadSupported: A Boolean that is TRUE if the volume supports the FSCTL_OFFLOAD_READ operation. This bit is reset to TRUE at mount time, and is set to FALSE if an Offload Read operation fails for an implementation- or vendor-specific reason.

IsOffloadWriteSupported: A Boolean that is TRUE if the volume supports the FSCTL_OFFLOAD_WRITE operation. This bit is reset to TRUE at mount time, and is set to FALSE if an Offload Write operation fails for an implementation- or vendor-specific reason.

MaxFileSize: A 64-bit unsigned integer that denotes the maximum file size, in bytes, supported by the object store.<10>

The following fields are specific to UDF object stores:

DirectoryCount: A 64-bit signed integer that indicates the count of directories on the volume, or -1 if not maintained by the object store.

FileCount: A 64-bit signed integer that indicates the count of files on the volume, or -1 if not maintained by the object store.

FsFormatMajVersion: A 16-bit unsigned integer indicating the major version of the file system format.

FsFormatMinVersion: A 16-bit unsigned integer indicating the minor version of the file system format.

FormatTime: The time the volume was formatted in the FILETIME format specified in [MS-FSCC] section 2.1.1.

LastUpdateTime: The time the volume was last updated in the FILETIME format specified in [MS-FSCC] section 2.1.1.

CopyrightInfo: A 68-byte buffer containing any copyright info associated with the volume.

AbstractInfo: A 68-byte buffer containing any abstract info associated with the volume.

FormattingImplementationInfo: A 68-byte buffer containing implementation-specific information; this field MAY contain the operating system version that the media was formatted by.

LastModifyingImplementationInfo: A 68-byte buffer containing information written by the last implementation that modified the disk. This field is implementation-specific and MAY contain the operating system version that the media was last modified by.

SparingUnitBytes: A 32-bit unsigned integer indicating the size in bytes of a sparing unit.

SoftwareSparing: A Boolean that is TRUE if the volume's bad block sparing mechanism is implemented in software, FALSE if bad block sparing is implemented by the underlying hardware this volume is on.

TotalSpareBlocks: A 32-bit unsigned integer indicating the total number of spare blocks.

FreeSpareBlocks: A 32-bit unsigned integer indicating the available number of spare blocks.

Volatile Fields:

OpenFileList: A list of all the **File** objects opened on **Volume**.

3.1.1.2 Per TunnelCacheEntry

Implementation of tunnel caching is optional.<11> If case-sensitive file name matching is enabled (for example, for POSIX compliance), the object store SHOULD NOT implement tunnel caching. If the object store implements tunnel caching, it MUST implement the following attributes in each **TunnelCacheEntry**:

EntryTime: The time at which this **TunnelCacheEntry** was created. The object store SHOULD use this attribute to automatically purge this entry from the tunnel cache once the entry is 15 seconds old.

ParentFile: The parent **DirectoryFile** that this **TunnelCacheEntry** refers to.

FileName: A Unicode string specifying the long name of the file. This string MUST be greater than 0 characters and less than 256 characters in length. Valid characters for a file name are specified in [MS-FSCC] section 2.1.5.

FileShortName: A Unicode string specifying the short name of the file. If **KeyByShortName** is FALSE, this string could be empty. If the string is not empty, it MUST be 8.3-compliant as described in [MS-FSCC] section 2.1.5.2.1.

KeyByShortName: A Boolean that is TRUE when **FileShortName** is used as the key for this entry. FALSE when **FileName** is used as the key for this entry.

FileCreationTime: The time that identifies when the file was created in the FILETIME format specified in [MS-FSCC] section 2.1.1.

FileObjectId: A GUID as specified in [RFC4122]. This value can be NULL. If non-NULL, this value MUST be unique on a given volume.

3.1.1.3 Per File

The object store MUST implement the following persistent attributes:

FileType: The type of file. This value MUST be either DataFile or DirectoryFile.

FileID: A 64-bit unsigned integer that identifies the file. This value MUST be persistent and MUST be unique on a given volume.

FileNumber: A 64-bit unsigned integer. This value MUST be persistent and MUST be unique on a given volume.

LinkList: A list of one or more **Links** to the file. A DirectoryFile MUST have exactly one element in **LinkList**. **LinkList** MUST have at most one element with a non-empty **ShortName**.<12>

SecurityDescriptor: The security descriptor for this file, in the format specified in [MS-DTYP] section 2.4.6.

FileAttributes: Attributes of the file in the form specified in [MS-FSCC] section 2.6.

CreationTime: The time that identifies when the file was created in the FILETIME format specified in [MS-FSCC] section 2.1.1.<13>

LastModificationTime: The time that identifies when the file contents were last modified in the FILETIME format specified in [MS-FSCC] section 2.1.1.<14>

LastChangeTime: The time that identifies when the file metadata or contents were last changed in the FILETIME format specified in [MS-FSCC] section 2.1.1.<15>

LastAccessTime: The time that identifies when the file was last accessed in the FILETIME format specified in [MS-FSCC] section 2.1.1. Updating this value when accesses occur is optional.<16><17>

ExtendedAttributes: A list of FILE_FULL_EA_INFORMATION structures as defined by MS-FSCC section 2.4.15.<18>

ExtendedAttributesLength: A 32-bit unsigned integer that contains the combined length of all the **ExtendedAttributes**. <19>

ObjectId: A GUID as specified in [RFC4122]. This value can be NULL. If set to non-NULL, this value MUST be unique on a given volume.<20>

BirthVolumeId: A GUID that uniquely identifies the volume on which the object resided when the object identifier was created, or zero if the volume had no object identifier at that time. After copy operations, move operations, or other file operations, this value is potentially different from the **VolumeId** of the volume on which the object currently resides.

BirthObjectId: A GUID value containing the object identifier of the object at the time it was created. After copy operations, move operations, or other file operations, this value is potentially different from the ObjectId member at present.<21>

StreamList: A list of zero or more **Streams** as defined in section 3.1.1.4. A DataFile MUST have one and only one unnamed DataStream; any additional streams MUST be named DataStreams.<22> A DirectoryFile MUST have one and only one unnamed DirectoryStream; any additional streams MUST be named DataStreams. For any two distinct elements *Stream1* and *Stream2* in **StreamList**, if *Stream1*.**StreamType** equals *Stream2*.**StreamType** then *Stream1*.**Name** MUST NOT match *Stream2*.**Name**.

ReparseTag: A 32-bit unsigned integer containing the type of the **reparse point**, as defined in [MS-FSCC] section 2.1.2.1. If this member is empty, there is no reparse point associated with this file.

ReparseGUID: A GUID indicating the type of the reparse point. This field MUST contain a valid GUID if **ReparseTag** contains a non-Microsoft tag as described in [MS-FSCC] section 2.1.2.1. Otherwise it MUST be empty.

ReparseData: An array of bytes containing data associated with a reparse point, which is defined by the type of the reparse point, as described in [MS-FSCC] sections 2.1.2.1 through 2.1.3.2. If ReparseTag is empty, this member MUST be empty. If ReparseTag is not empty, this member could be empty, in which case there is no reparse data associated with this reparse point.

DirectoryList: For a DataFile, this list MUST be empty. For a DirectoryFile, this is a list of **Links** contained in the directory. For any two distinct elements *Link1* and *Link2* in **DirectoryList**, *Link1*.**Name** MUST NOT match *Link2*.**Name** or *Link2*.**ShortName**.<23>

Volume: The **Volume** on which the file resides.

Usn: A 64-bit unsigned integer indicating the positive USN number of the last USN record written for this file, or 0 if no USN records have been written for this file.

IsSymbolicLink: A Boolean that is TRUE if the file is a **mount point** or a **symbolic link** to another file or directory.

UserCertificateList: A list of **ENCRYPTION_CERTIFICATE** structures as specified in [MS-EFSR] section 2.2.8, used to determine which users can access the contents of any encrypted streams in the file.<24>

Volatile Fields:

OpenList: A list of all **Opens** to this **File**.

PendingNotifications: A 32-bit unsigned integer composed of flags indicating types of changes to file attributes for which directory change notifications are pending, as specified in [MS-SMB2] section 2.2.35, **CompletionFilter** field.

3.1.1.4 Per Link

The object store MUST implement the following persistent attributes:<25>

Name: A Unicode string specifying the name of the link. This string MUST be greater than 0 characters and less than 256 characters in length. Valid form for a link name is the same as the pathname specification in [MS-FSCC] section 2.1.5.

ShortName: A Unicode string specifying the short name of the link.<26> This value could be empty. If this value is not empty, it MUST be 8.3-compliant as described in [MS-FSCC] section 2.1.5.2.1.

File: The **File** that this link refers to.

ParentFile: The parent **DirectoryFile** that this link resides in.

IsDeleted: A Boolean that is TRUE if there is a pending delete operation on the link. New opens to the associated Stream MUST NOT be allowed.

Volatile Fields:

PendingNotifications: A 32-bit unsigned integer composed of flags indicating types of changes to link attributes for which directory change notifications are pending, as specified in [MS-SMB2] section 2.2.35, **CompletionFilter** field.

3.1.1.5 Per Stream

Note: Some of the information in this section is subject to change because it applies to a preliminary implementation of the protocol or structure. For information about specific differences between versions, see the behavior notes that are provided in the Product Behavior appendix.

The object store MUST implement the following persistent attributes:

StreamType: The type of stream. This value MUST be either DataStream or DirectoryStream.

Name: A Unicode string of less than 256 characters specifying the name of the stream. Valid characters for a stream name are specified in [MS-FSCC] section 2.1.5. If **StreamType** is DataStream, **Name** could be empty; this case indicates the **default data stream**. If **StreamType** is DirectoryStream, **Name** MUST be empty.

Size: A 64-bit unsigned integer containing the size of the stream, in bytes.

AllocationSize: A 64-bit unsigned integer containing the size, in bytes, of space reserved on the disk. This value MUST be a multiple of **File.Volume.ClusterSize**.

ValidDataLength: A 64-bit unsigned integer containing the size, in bytes, of valid data in the stream. Not all file systems support this field, and implementation of this field is optional. If implemented, all data beyond this value MUST be returned as zero. For a DataStream, this value MUST be less than or equal to **Size**. For a DirectoryStream, this value MUST be equal to **Size**.

File: The **File** in which the stream resides.

IsCompressed: A Boolean that is TRUE if the contents of the stream are compressed.<27>

IsIntegrity: A Boolean that is TRUE if the contents of the stream have integrity.<28>

IsChecksumEnforcementOff: A Boolean that is TRUE if the stream is a FileStream and CHECKSUM_ENFORCEMENT_OFF is specified.<29>

IsSparse: A Boolean that is TRUE if the object store is storing a sparse representation of the stream.<30>

IsTemporary: A Boolean that is TRUE if the object store optimizes its management of the stream because it is pending deletion.

IsEncrypted: A Boolean that is TRUE if the contents of the stream are encrypted.<31>

ExtentList: A list containing zero or more EXTENTS elements as defined by [MS-FSCC] section 2.3.20.1, ordered by **NextVcn**.

Volatile Fields:

Oplock: An **Oplock** describing the opportunistic lock state of the stream. If **Oplock** is empty, there is no opportunistic lock on the stream.

ByteRangeLockList: A list of zero or more **ByteRangeLocks** describing the bytes ranges of this stream that are currently locked.

IsDeleted: A Boolean that is TRUE if there is a pending delete operation on the **Stream**. New opens to **Stream** MUST NOT be allowed.

IsDefectManagementDisabled: A Boolean that is TRUE if **software defect management** is disabled on this stream. Not all file systems support this field; implementation of this field is optional.

PendingNotifications: A 32-bit unsigned integer composed of flags indicating types of changes to stream attributes for which directory change notifications are pending, as specified in [MS-SMB2] section 2.2.35, **CompletionFilter** field.

ZeroOnDeallocate: A Boolean that is TRUE when the object store MUST write zeroes to any range of the stream that is to be deallocated, prior to performing the deallocation. This helps to protect whatever data may have been in the stream from discovery by examining free space on the storage media. Not all file systems support this field, and implementation of this field is optional.

3.1.1.6 Per Open

The object store MUST implement the following:

RootOpen: The **Open** that represents the root of the share.

FileName: The absolute pathname of the opened file in the format specified in [MS-FSCC] section 2.1.5.

File: The **File** that is opened.

Link: The **Link** through which **File** is opened. **Link** MUST be an element of **File.LinkList**.

Stream: The **Stream** that is opened. **Stream** MUST be an element of **File.StreamList**.

GrantedAccess: The access granted for this open as specified in [MS-SMB2] section 2.2.13.1.

RemainingDesiredAccess: The access requested for this Open but not yet granted, as specified in [MS-SMB2] section 2.2.13.1.

SharingMode: The sharing mode for this Open as specified in [MS-SMB2] section 2.2.13.

Mode: The mode flags for this Open as specified in [MS-FSCC] section 2.4.24.

IsCaseInsensitive: A Boolean that is TRUE if this Open should be treated as case-insensitive.

HasBackupAccess: A Boolean that is TRUE if the Open was performed by a user who is allowed to perform **backup** operations.

HasRestoreAccess: A Boolean that is TRUE if the Open was performed by a user who is allowed to perform **restore** operations.

HasCreateSymbolicLinkAccess: A Boolean that is TRUE if the Open was performed by a user who is allowed to create symbolic links.

HasManageVolumeAccess: A Boolean that is TRUE if the Open was performed by a user who is allowed to manage the volume.

IsAdministrator: A Boolean that is TRUE if the Open was performed by a user who is a member of the BUILTIN_ADMINISTRATORS group as specified in [MS-DTYP] section 2.4.2.4.

QueryPattern: The Unicode string containing the query pattern used to filter directory query.

QueryLastEntry: The last **Link** that was returned in a directory query.

LastQuotaId: The index of the last SID returned during quota enumeration on this Open, or -1 if there has not been a quota enumeration on this Open.

CurrentByteOffset: The byte offset immediately following the most recent successful synchronous read or write operation of one or more bytes, or 0 if there have not been any.

FindBySidRestartIndex: A 64-bit unsigned integer specifying the starting index for a FSCTL_FILE_FILES_BY_SID operation.

UserSetModificationTime: A Boolean that is TRUE if a user has explicitly set **File.LastModificationTime** through this Open.

UserSetChangeTime: A Boolean that is TRUE if a user has explicitly set **File.LastChangeTime** through this Open.

UserSetAccessTime: A Boolean that is TRUE if a user has explicitly set **File.LastAccessTime** through this Open.

NextEaEntry: Contains a reference to the next FILE_FULL_EA_INFORMATION entry in **File.ExtendedAttributes** to be returned the next time FileFullEaInformation is called using this Open as defined in section 3.1.5.11.12.<32>

TargetOplockKey: A GUID value that may be used to identify the owner of the **Open** for the purpose of determining whether to break an oplock in response to a request delivered on a particular **Open**. Requests on an **Open** whose **Open.TargetOplockKey** value matches the **Open.TargetOplockKey** value associated with an oplock that exists on the **Stream** do not affect the oplock state (that is, do not cause the oplock to break). For a given **Open**, the **TargetOplockKey** value could be empty. An empty value MUST NOT be considered equal to anything other than itself. In other words, given two **Open** values, *Open1* and *Open2*, such that

Open1.**TargetOplockKey** and/or *Open2*.**TargetOplockKey** are empty, *Open1*.**TargetOplockKey** MUST NOT be considered equal to *Open2*.**TargetOplockKey**.

ParentOplockKey: A GUID value that can be used to identify the owner of an oplock on the parent directory of the **File** associated with the current **Open** for the purpose of determining whether to break an oplock on the parent in response to a request delivered on a particular **Open** to a child of that parent. Requests on an **Open** whose **Open.ParentOplockKey** value matches the **Open.TargetOplockKey** value associated with an oplock that exists on the parent directory **Stream** do not affect the parent's oplock state (that is, do not cause the oplock to break). For a given **Open**, the **TargetOplockKey** value could be empty. An empty value MUST NOT be considered equal to anything other than itself. In other words, given two **Open** values, *ParentOpen* on a directory and *ChildOpen* on a child (either file or directory), such that *ParentOpen*.**TargetOplockKey** and/or *ChildOpen*.**ParentOplockKey** are empty, *ParentOpen*.**TargetOplockKey** MUST NOT be considered equal to *ChildOpen*.**ParentOplockKey**.

3.1.1.7 Per ByteRangeLock

LockOffset: A 64-bit unsigned integer specifying the offset, in bytes, from the beginning of a stream where the locked range begins.

LockLength: A 64-bit unsigned integer specifying the length, in bytes, of the locked range.

IsExclusive: A Boolean that is TRUE if this is an exclusive byte range lock, else FALSE if this is a shared byte range lock.

OwnerOpen: The **Open** that owns this **ByteRangeLock**.

3.1.1.8 Per ChangeNotifyEntry

OpenedDirectory: The **Open** of the **DirectoryFile** to monitor for changes.

WatchTree: A Boolean value, set to TRUE if changes to subdirectories MUST be notified, FALSE if not.

CompletionFilter: A 32-bit unsigned integer composed of flags indicating the types of changes to monitor as specified in [MS-SMB2] section 2.2.35.

NotifyEventList: A list of **NotifyEventEntries**, representing change events that were not yet reported to the user.

3.1.1.9 Per NotifyEventEntry

Action: A 32-bit unsigned integer composed of flags indicating the type of change events that occurred, as specified in [MS-SMB2] section 2.2.36.1.

FileName: Pathname relative to **ChangeNotifyEntry.OpenedDirectory** of the file involved in the change event.

3.1.1.10 Per Oplock

ExclusiveOpen: The **Open** used to request the opportunistic lock.

IIOplocks: A list of zero or more **Opens** used to request a LEVEL_TWO opportunistic lock, as specified in section 3.1.5.17.1.

ROplocks: A list of zero or more **Opens** used to request a LEVEL_GRANULAR(**RequestedOplockLevel**: READ_CACHING) opportunistic lock, as specified in section 3.1.5.17.1.

RHOplocks: A list of zero or more **Opens** used to request a LEVEL_GRANULAR(**RequestedOplockLevel**: (READ_CACHING|HANDLE_CACHING)) opportunistic lock, as specified in section 3.1.5.17.1.

RHBreakQueue: A list of zero or more **RHOpContext** objects. This queue is used to track (READ_CACHING|HANDLE_CACHING) oplocks as they are breaking.

WaitList: A list of zero or more **Opens** belonging to operations that are waiting for an oplock to break, as specified in section 3.1.4.12.

State: The current state of the oplock, expressed as a combination of one or more flags. Valid flags are:

NO_OPLOCK - Indicates that this **Oplock** does not represent a currently granted or breaking oplock. This is semantically equivalent to the **Oplock** object being entirely absent from a **Stream**. This flag always appears alone.

LEVEL_ONE_OPLOCK - Indicates that this **Oplock** represents a Level 1 (also called Exclusive) oplock.

BATCH_OPLOCK - Indicates that this **Oplock** represents a Batch oplock.

LEVEL_TWO_OPLOCK - Indicates that this **Oplock** represents a Level 2 (also called Shared) oplock.

EXCLUSIVE - Indicates that this **Oplock** represents an oplock that can be held by exactly one client at a time. This flag always appears in combination with other flags that indicate the actual oplock level. For example, (READ_CACHING|WRITE_CACHING|EXCLUSIVE) represents a read caching and write caching oplock, which can be held by only one client at a time.

BREAK_TO_TWO - Indicates that this **Oplock** represents an oplock that is currently breaking from either Level 1 or Batch to Level 2; the oplock has broken but the break has not yet been acknowledged.

BREAK_TO_NONE - Indicates that this **Oplock** represents an oplock that is currently breaking from either Level 1 or Batch to None (that is, no oplock); the oplock has broken but the break has not yet been acknowledged.

BREAK_TO_TWO_TO_NONE - Indicates that this **Oplock** represents an oplock that is currently breaking from either Level 1 or Batch to None (that is, no oplock), and was previously breaking from Level 1 or Batch to Level 2; the oplock has broken but the break has not yet been acknowledged.

READ_CACHING - Indicates that this **Oplock** represents an oplock that provides caching of reads; this provides the SMB 2.1 read caching lease, as described in [MS-SMB2] section 2.2.13.2.8.

HANDLE_CACHING - Indicates that this **Oplock** represents an oplock that provides caching of handles; this provides the SMB 2.1 handle caching lease, as described in [MS-SMB2] section 2.2.13.2.8.

WRITE_CACHING - Indicates that this **Oplock** represents an oplock that provides caching of writes; this provides the SMB 2.1 write caching lease, as described in [MS-SMB2] section 2.2.13.2.8.

MIXED_R_AND_RH - Always appears together with READ_CACHING and HANDLE_CACHING. Indicates that this **Oplock** represents an oplock on which at least one client has been granted a read caching oplock, and at least one other client has been granted a read caching and handle caching oplock.

BREAK_TO_ READ_CACHING - Indicates that this **Oplock** represents an oplock that is currently breaking to an oplock that provides caching of reads; the oplock has broken but the break has not yet been acknowledged.

BREAK_TO_WRITE_CACHING - Indicates that this **Oplock** represents an oplock that is currently breaking to an oplock that provides caching of writes; the oplock has broken but the break has not yet been acknowledged.

BREAK_TO_HANDLE_CACHING - Indicates that this **Oplock** represents an oplock that is currently breaking to an oplock that provides caching of handles; the oplock has broken but the break has not yet been acknowledged.

BREAK_TO_NO_CACHING - Indicates that this **Oplock** represents an oplock that is currently breaking to None (that is, no oplock); the oplock has broken but the break has not yet been acknowledged.

3.1.1.11 Per RHOpContext

Open: The **Open** used to request this LEVEL_GRANULAR(**RequestedOplockLevel**: (READ_CACHING|HANDLE_CACHING)) opportunistic lock.

BreakingToRead: A Boolean value that is TRUE if this oplock is breaking to READ_CACHING, FALSE if it is breaking to None (that is, no oplock; the oplock is being broken completely).

3.1.1.12 Per CancelableOperations

CancelableOperationList: A global list of cancelable operations currently being processed by the object store. Items in this list are looked up via their **IORequest** Identifier as defined in section 3.1.5.19. Operations are inserted into this list when a cancelable operation waits.

3.1.1.13 Per SecurityContext

SIDs: An array of SID structures, as specified in [MS-DTYP] section 2.4.2, representing the security identifier of the user performing an operation and the security identifiers of all groups of which the user is a member.

OwnerIndex: An index into **SIDs** indicating the SID of the user.

PrimaryGroup: An index into **SIDs** indicating the SID of the user's primary group.

DefaultDACL: An ACL structure, as specified in [MS-DTYP] section 2.4.5, representing the default DACL assigned to new files created by the user.

PrivilegeSet: A set of privilege names, as specified in [MS-LSAD] section 3.1.1.2.1, representing the privileges held by the user.

3.1.2 Timers

The object store has no timers.

3.1.3 Initialization

On initialization, one or more **Volume** objects are initialized based on the data stored in the persistent store. This involves instantiating one or more **File** objects contained within the volume.

3.1.4 Common Algorithms

This section describes internal algorithms that are common across multiple triggered events.

3.1.4.1 Algorithm for Reporting a Change Notification for a Directory

The inputs for this algorithm are:

Volume: The volume this event occurs on.

Action: A 32-bit unsigned integer describing the action that caused the change events to be notified, as specified in [MS-SMB2] section 2.2.36.1.

FilterMatch: A 32-bit unsigned integer field with flags representing possible change events, corresponding to a **ChangeNotifyEntry.CompletionFilter**. It is specified in [MS-SMB2] section 2.2.35.

FileName: The pathname, relative to **Volume.RootDirectory**, of the file involved in the change event.

Pseudocode for the algorithm is as follows:

For each **ChangeNotifyEntry** in **Volume.ChangeNotifyList**:

Initialize *SendNotification* to FALSE.

If **ChangeNotifyEntry.OpenedDirectory.File** matches the **File** whose pathname is **FileName** or matches the immediate parent of this **File** and one or more of the flags in **FilterMatch** are present in **ChangeNotifyEntry.CompletionFilter**, then *SendNotification* MUST be set to TRUE.

Else If **ChangeNotifyEntry.WatchTree** is TRUE and **ChangeNotifyEntry.OpenedDirectory.File** matches an ancestor of the **File** whose pathname is **FileName** and one or more of the flags in **FilterMatch** are present in **ChangeNotifyEntry.CompletionFilter**, then *SendNotification* MUST be set to TRUE.

EndIf

If *SendNotification* is TRUE:

A **NotifyEventEntry** object MUST be constructed with:

NotifyEventEntry.Action set to **Action**.

NotifyEventEntry.FileName set to the portion of **FileName** relative to **ChangeNotifyEntry.OpenedDirectory.FileName**.

Insert **NotifyEventEntry** into **ChangeNotifyEntry.NotifyEventList**.

Processing will be performed as described in section 3.1.5.10.1.

EndIf

EndFor

3.1.4.2 Algorithm for Detecting If Open Files Exist Within a Directory

Return FALSE // An open child still exists, deny the operation.

EndIf

EndFor

Return TRUE // No opens remaining.

3.1.4.3 Algorithm for Determining If a Character Is a Wildcard

The following set of characters MUST be treated as wildcards by the object store:

" * < > ?

3.1.4.4 Algorithm for Determining if a FileName Is in an Expression

The inputs for this algorithm are:

FileName: A Unicode string containing the file name string that is being matched. **Filename** may not contain any wildcard characters.

Expression: A Unicode string containing the regular expression that's being matched with **FileName**.

IgnoreCase: A Boolean value indicating whether the match is case insensitive (TRUE) or case sensitive (FALSE).

This algorithm returns TRUE if **FileName** matches **Expression**, and FALSE if it does not.

Pseudocode for the algorithm is as follows:

Part 1 -- Handle Special Case Optimizations

If **FileName** is empty and **Expression** is not, the routine returns FALSE.

If **Expression** is empty and **FileName** is not, the routine returns FALSE.

If both **Expression** and **FileName** are empty, the routine returns TRUE.

If the **Expression** is the wildcard "*" or "*.*", the **FileName** matches the **Expression** and the routine returns TRUE.

If the first character in the **Expression** is wildcard "*" and the rest of the expression does not contain any wildcard characters (as per 3.1.4.3), then the remaining expression is compared against the tail end of the **FileName**. If the comparison succeeds then the routine returns TRUE.

Part 2 -- Match Expression with FileName

The **FileName** is string compared with **Expression** using the following wildcard rules:

* (asterisk) Matches zero or more characters.

? (question mark) Matches a single character.

DOS_DOT (" quotation mark) Matches either a period or zero characters beyond the name string.

DOS_QM (> greater than) Matches any single character or, upon encountering a period or end of name string, advances the expression to the end of the set of contiguous DOS_QMs.

DOS_STAR (< less than) Matches zero or more characters until encountering and matching the final . in the name.

3.1.4.5 BlockAlign -- Macro to Round a Value Up to the Next Nearest Multiple of Another Value

The inputs for this algorithm are:

Value: The value being rounded up.

Boundary - **Value** is to be rounded up to a multiple of this value. **Boundary** MUST be a power of 2.

This algorithm returns the bitwise AND of (**Value** + (**Boundary** - 1)) with the 2's complement of **Boundary**.

Pseudocode for the algorithm is as follows:

BlockAlign(**Value**, **Boundary**) = (**Value** + (**Boundary** - 1)) & -(**Boundary**)

3.1.4.6 BlockAlignTruncate -- Macro to Round a Value Down to the Next Nearest Multiple of Another Value

The inputs for this algorithm are:

Value: The value being rounded down.

Boundary - **Value** is to be rounded down to a multiple of this value.**Boundary** MUST be a power of 2.

This algorithm returns the bitwise AND of **Value** with the 2's complement of **Boundary**.

Pseudocode for the algorithm is as follows:

BlockAlignTruncate(**Value**, **Boundary**) = **Value** & -(**Boundary**)

3.1.4.7 ClustersFromBytes -- Macro to Determine How Many Clusters a Given Number of Bytes Occupies

The inputs for this algorithm are:

ThisVolume: A **Volume**.

Bytes: The number of bytes.

Pseudocode for the algorithm is as follows:

ClustersFromBytes(**ThisVolume**, **Bytes**) = (**Bytes** + (**ThisVolume.ClusterSize** - 1)) / **ThisVolume.ClusterSize**.

The value returned is the total number of clusters required to hold the specified number of bytes that start at a cluster boundary, including any remainder that does not fill a whole cluster.

3.1.4.8 ClustersFromBytesTruncate -- Macro to Determine How Many Whole Clusters a Given Number of Bytes Occupies

The inputs for this algorithm are:

ThisVolume: A **Volume**.

Bytes: The number of bytes.

Pseudocode for the algorithm is as follows:

ClustersFromBytesTruncate(**ThisVolume**, **Bytes**) = **Bytes** / **ThisVolume.ClusterSize**.

The value returned is the number of clusters that would be fully occupied by the specified number of bytes that start at a cluster boundary. Any remainder that does not fill a whole cluster is discarded.

3.1.4.9 SidLength -- Macro to Provide the Length of a SID

The inputs for this algorithm are:

SID: A SID, as described in [MS-DTYP] section 2.4.2.

This algorithm returns the size, in bytes, of **SID**. This is equal to the number of bytes occupied by the **Revision**, **SubAuthorityCount**, and **IdentifierAuthorityCount** fields of a SID. Added to this is the size of a **SubAuthority** field of a SID times **SID.SubAuthorityCount**.

Pseudocode for the algorithm is as follows:

SidLength(SID) = (8 + (4 * SID.SubAuthorityCount))

3.1.4.10 Algorithm for Determining If a Range Access Conflicts with Byte-Range Locks

The inputs for this algorithm are:

ByteOffset: A 64-bit unsigned integer specifying the offset of the first byte of the range.

Length: A 64-bit unsigned integer specifying the number of bytes in the range.

IsExclusive: TRUE if the access to the range has exclusive intent, FALSE otherwise.

LockIntent: TRUE if the access to the range has locking intent, FALSE if the intent is performing I/O (reads or writes).

Open: The open to the file on which to check for range conflicts.

This algorithm outputs a Boolean value:

TRUE if the range conflicts with byte-range locks.

FALSE if the range does not conflict.

Pseudocode for the algorithm is as follows:

If ((**ByteOffset** == 0) and (**Length** == 0)):

The {0, 0} range doesn't conflict with any byte-range lock.

Return FALSE.

EndIf

For each *ByteRangeLock* in **Open.Stream.ByteRangeLockList**:

If ((*ByteRangeLock*.**LockOffset** == 0) and (*ByteRangeLock*.**LockLength** == 0)):

The byte-range lock is over the {0, 0} range so there is no overlap by definition.

Else:

Initialize *LastByteOffset1* = **ByteOffset** + **Length** - 1.

Initialize *LastByteOffset2* = *ByteRangeLock*.**LockOffset** + *ByteRangeLock*.**LockLength** - 1.

If ((**ByteOffset** <= *LastByteOffset2*) and (*LastByteOffset1* >= *ByteRangeLock*.**LockOffset**)):

ByteRangeLock and the passed range overlap.

If (*ByteRangeLock*.**IsExclusive** == TRUE):

If (*ByteRangeLock*.**OwnerOpen** != **Open**):

Exclusive byte-range locks block all access to other **Opens**.

Return TRUE.

Else If ((**IsExclusive** == TRUE) and (**LockIntent** == TRUE)):

Overlapping exclusive byte-range locks are not allowed even by the same owner.

Return TRUE.

EndIf

Else If (**IsExclusive** == TRUE):

The *ByteRangeLock* is shared, shared byte-range locks will block all access with exclusive intent.

Return TRUE.

EndIf

EndIf

EndIf

EndFor

Return FALSE.

3.1.4.11 Algorithm for Posting a USN Change for a File

The inputs for this algorithm are:

File: The file this change occurs on.

Reason: A 32-bit unsigned integer describing the change that occurred to the file, as specified in [MS-FSCC] section 2.3.40.

FileName: The pathname, relative to **Volume.RootDirectory**, of the file this change occurs on.

The algorithm MUST return at this point without taking any actions under any of the following conditions:

If the object store does not support USN change journals.

If **File.Volume.IsUsnJournalActive** is FALSE.

If **Reason** is zero.

Pseudocode for the algorithm is as follows:

Set *FileNameLength* to the length, in bytes, of **FileName**.

Set *RecordLength* to *BlockAlign**(**FieldOffset**(*USN_RECORD.FileName***)** + *FileNameLength*, 8**)**.

Set **File.Volume.LastUsn** to **File.Volume.LastUsn** + *RecordLength*.

Set **File.Usn** to **File.Volume.LastUsn**.

3.1.4.12 Algorithm to Check for an Oplock Break

Note: Some of the information in this section is subject to change because it applies to a preliminary implementation of the protocol or structure. For information about specific differences between versions, see the behavior notes that are provided in the Product Behavior appendix.

The inputs for this algorithm are:

Open: The **Open** being used in the request calling this algorithm.

Oplock: The **Oplock** being checked.

Operation: A code describing the operation being processed.

OpParams: Parameters associated with the **Operation** code that are passed in from the calling request. For example, if **Operation** is OPEN, as specified in section 3.1.5.1, then **OpParams** will have the members **DesiredAccess** and **CreateDisposition**. Each of these is a parameter to the open request as specified in section 3.1.5.1. This parameter could be empty, depending on the **Operation** code.

Flags: An optional parameter. If unspecified it is considered to contain 0. Valid nonzero values are:

PARENT_OBJECT

The algorithm uses the following local variables:

Boolean values (initialized to FALSE): *BreakToTwo*, *BreakToNone*, *NeedToWait*

BreakCacheLevel – MAY contain 0 or a combination of one or more of READ_CACHING, WRITE_CACHING, or HANDLE_CACHING, as specified in section 3.1.1.10. Initialized to 0.

Note that there are only four legal nonzero combinations of flags for *BreakCacheLevel*:

(READ_CACHING|WRITE_CACHING|HANDLE_CACHING)

(READ_CACHING|WRITE_CACHING)

WRITE_CACHING

HANDLE_CACHING

Pseudocode for the algorithm is as follows:

If **Oplock** is not empty and **Oplock.State** is not NO_OPLOCK:

If **Flags** contains PARENT_OBJECT:

If **Operation** is OPEN, as specified in section 3.1.5.1, or

Operation is FLUSH_DATA, as specified in section 3.1.5.6, or

Operation is CLOSE, as specified in section 3.1.5.4, or

Operation is FS_CONTROL, as specified in section 3.1.5.9, and **OpParams.ControlCode** is FSCTL_SET_ENCRYPTION, or

Operation is SET_INFORMATION, as specified in section 3.1.5.14, and **OpParams.FileInformationClass** is one of FileBasicInformation or FileAllocationInformation or FileEndOfFileInformation or FileRenameInformation or FileLinkInformation or FileShortNameInformation:

Set *BreakCacheLevel* to (READ_CACHING|WRITE_CACHING).

EndIf

Else:

Switch (**Operation**):

Case OPEN, as specified in section 3.1.5.1:

If **OpParams.DesiredAccess** contains no flags other than FILE_READ_ATTRIBUTES, FILE_WRITE_ATTRIBUTES, or SYNCHRONIZE, the algorithm returns at this point.

EndIf

If **OpParams.CreateDisposition** is FILE_SUPERSEDE, FILE_OVERWRITE, or FILE_OVERWRITE_IF:

Set *BreakToNone* to TRUE, set *BreakCacheLevel* to (READ_CACHING|WRITE_CACHING).

Else

Set *BreakToTwo* to TRUE, set *BreakCacheLevel* to WRITE_CACHING.

EndIf

EndCase

Case OPEN_BREAK_H, as specified in section 3.1.5.1:

Set *BreakCacheLevel* to HANDLE_CACHING.

EndCase

Case CLOSE, as specified in section 3.1.5.4:

If **Oplock.IIOplocks** is not empty:

For each **Open** *ThisOpen* in **Oplock.IIOplocks**:

If *ThisOpen* == **Open**:

Remove *ThisOpen* from **Oplock.IIOplocks**.

Notify the **server** of an oplock break according to the algorithm in section 3.1.5.17.3, setting the algorithm's parameters as follows:

BreakingOplockOpen equal to *ThisOpen*.

NewOplockLevel equal to LEVEL_NONE.

AcknowledgeRequired equal to FALSE.

OplockCompletionStatus equal to STATUS_SUCCESS.

(The operation does not end at this point; this call to 3.1.5.17.3 completes some earlier call to 3.1.5.17.2.)

EndIf

EndFor

Recompute **Oplock.State** according to the algorithm in section 3.1.4.13, passing **Oplock** as the **ThisOplock** parameter.

EndIf

If **Oplock.ROplocks** is not empty:

For each **Open** *ThisOpen* in **Oplock.ROplocks**:

If *ThisOpen* == **Open**:

Remove *ThisOpen* from **Oplock.ROplocks**.

Notify the server of an oplock break according to the algorithm in section 3.1.5.17.3, setting the algorithm's parameters as follows:

BreakingOplockOpen equal to *ThisOpen*.

NewOplockLevel equal to LEVEL_NONE.

AcknowledgeRequired equal to FALSE.

OplockCompletionStatus equal to STATUS_OPLOCK_HANDLE_CLOSED.

(The operation does not end at this point; this call to 3.1.5.17.3 completes some earlier call to 3.1.5.17.2.)

EndIf

EndFor

Recompute **Oplock.State** according to the algorithm in section 3.1.4.13, passing **Oplock** as the **ThisOplock** parameter.

EndIf

If **Oplock.RHOplocks** is not empty:

For each **Open** *ThisOpen* in **Oplock.RHOplocks**:

If *ThisOpen* == **Open**:

Remove *ThisOpen* from **Oplock.RHOplocks**.

Notify the server of an oplock break according to the algorithm in section 3.1.5.17.3, setting the algorithm's parameters as follows:

BreakingOplockOpen equal to *ThisOpen*.

NewOplockLevel equal to LEVEL_NONE.

AcknowledgeRequired equal to FALSE.

OplockCompletionStatus equal to STATUS_OPLOCK_HANDLE_CLOSED.

(The operation does not end at this point; this call to 3.1.5.17.3 completes some earlier call to 3.1.5.17.2.)

EndIf

EndFor

Recompute **Oplock.State** according to the algorithm in section 3.1.4.13, passing **Oplock** as the **ThisOplock** parameter.

EndIf

If **Oplock.RHBreakQueue** is not empty:

For each **RHOpContext** *ThisContext* in **Oplock.RHBreakQueue**:

If *ThisContext*.**Open** == **Open**:

Remove *ThisContext* from **Oplock.RHBreakQueue**.

EndIf

EndFor

Recompute **Oplock.State** according to the algorithm in section 3.1.4.13, passing **Oplock** as the **ThisOplock** parameter.

For each **Open** *WaitingOpen* on **Oplock.WaitList**:

If **Oplock.RHBreakQueue** is empty:

Indicate that the operation associated with *WaitingOpen* may continue according to the algorithm in section 3.1.4.12.1, setting **OpenToRelease** equal to *WaitingOpen*.

Remove *WaitingOpen* from **Oplock.WaitList**.

Else

If the value on every **RHOpContext.Open.TargetOplockKey** on **Oplock.RHBreakQueue** is equal to *WaitingOpen* **.TargetOplockKey**:

Indicate that the operation associated with *WaitingOpen* may continue according to the algorithm in section 3.1.4.12.1, setting **OpenToRelease** equal to *WaitingOpen*.

Remove *WaitingOpen* from **Oplock.WaitList**.

EndIf

EndIf

EndFor

EndIf

If **Open** equals **Open.Oplock.ExclusiveOpen**

If **Oplock.State** contains none of BREAK_TO_TWO, BREAK_TO_NONE, BREAK_TO_TWO_TO_NONE, BREAK_TO_READ_CACHING, BREAK_TO_WRITE_CACHING, BREAK_TO_HANDLE_CACHING, or BREAK_TO_NO_CACHING:

Notify the server of an oplock break according to the algorithm in section 3.1.5.17.3, setting the algorithm's parameters as follows:

BreakingOplockOpen equal to **Oplock.ExclusiveOpen**.

NewOplockLevel equal to LEVEL_NONE.

AcknowledgeRequired equal to FALSE.

OplockCompletionStatus equal to:

STATUS_OPLOCK_HANDLE_CLOSED if **Oplock.State** contains any of READ_CACHING, WRITE_CACHING, or HANDLE_CACHING.

STATUS_SUCCESS otherwise.

(The operation does not end at this point; this call to 3.1.5.17.3 completes some earlier call to 3.1.5.17.1.)

EndIf

Set **Oplock.ExclusiveOpen** to NULL.

Set **Oplock.State** to NO_OPLOCK.

For each **Open** *WaitingOpen* on **Oplock.WaitList**:

Indicate that the operation associated with *WaitingOpen* may continue according to the algorithm in section 3.1.4.12.1, setting **OpenToRelease** equal to *WaitingOpen*.

Remove *WaitingOpen* from **Oplock.WaitList**.

EndFor

EndIf

EndCase

Case READ, as specified in section 3.1.5.2:

Set *BreakToTwo* to TRUE

Set *BreakCacheLevel* to WRITE_CACHING.

EndCase

Case FLUSH_DATA, as specified in section 3.1.5.6:

Set *BreakToTwo* to TRUE

Set *BreakCacheLevel* to WRITE_CACHING.

EndCase

Case LOCK_CONTROL, as specified in section 3.1.5.7:

Case WRITE, as specified in section 3.1.5.3:

Set *BreakToNone* to TRUE

Set *BreakCacheLevel* to (READ_CACHING|WRITE_CACHING).

EndCase

Case SET_INFORMATION, as specified in section 3.1.5.14:

Switch (**OpParams.FileInformationClass**):

Case FileEndOfFileInformation:

Case FileAllocationInformation:

Set *BreakToNone* to TRUE

Set *BreakCacheLevel* to (READ_CACHING|WRITE_CACHING).

EndCase

Case FileRenameInformation:

Case FileLinkInformation:

Case FileShortNameInformation:

Set *BreakCacheLevel* to HANDLE_CACHING.

If **Oplock.State** contains BATCH_OPLOCK, set *BreakToNone* to TRUE.

EndCase

Case FileDispositionInformation:

If **OpParams.DeleteFile** is TRUE,

Set *BreakCacheLevel* to HANDLE_CACHING.

EndCase

EndSwitch // FileInfoClass

Case FS_CONTROL, as specified in section 3.1.5.9:

If **OpParams.ControlCode** is FSCTL_SET_ZERO_DATA:

Set *BreakToNone* to TRUE.

Set *BreakCacheLevel* to (READ_CACHING|WRITE_CACHING).

EndIf

EndCase

EndSwitch // **Operation**

EndIf

If *BreakToTwo* is TRUE:

If (**Oplock.State** != LEVEL_TWO_OPLOCK) and

((**Oplock.ExclusiveOpen** is empty) or

(**Oplock.ExclusiveOpen.TargetOplockKey** != **Open.TargetOplockKey**)):

If (**Oplock.State** contains EXCLUSIVE) and

(**Oplock.State** contains none of READ_CACHING, WRITE_CACHING, or HANDLE_CACHING):

If **Oplock.State** contains none of BREAK_TO_TWO, BREAK_TO_NONE, BREAK_TO_TWO_TO_NONE, BREAK_TO_READ_CACHING, BREAK_TO_WRITE_CACHING, BREAK_TO_HANDLE_CACHING, or BREAK_TO_NO_CACHING:

// **Oplock.State** MUST contain either LEVEL_ONE_OPLOCK or BATCH_OPLOCK.

Set BREAK_TO_TWO in **Oplock.State**.

Notify the server of an oplock break according to the algorithm in section 3.1.5.17.3, setting the algorithm's parameters as follows:

BreakingOplockOpen equal to **Oplock.ExclusiveOpen**.

NewOplockLevel equal to LEVEL_TWO.

AcknowledgeRequired equal to TRUE.

OplockCompletionStatus equal to STATUS_SUCCESS.

(The operation does not end at this point; this call to 3.1.5.17.3 completes some earlier call to 3.1.5.17.1.)

EndIf

The operation that called this algorithm MUST be made cancelable by inserting it into **CancelableOperations.CancelableOperationList**.

The operation that called this algorithm waits until the oplock break is acknowledged, as specified in section 3.1.5.18, or the operation is canceled.

EndIf

EndIf

Else If *BreakToNone* is TRUE:

If (**Oplock.State** == LEVEL_TWO_OPLOCK) or

(**Oplock.ExclusiveOpen** is empty) or

(**Oplock.ExclusiveOpen.TargetOplockKey** != **Open.TargetOplockKey**):

If (**Oplock.State** != NO_OPLOCK) and

(**Oplock.State** contains neither WRITE_CACHING nor HANDLE_CACHING):

If **Oplock.State** contains none of LEVEL_TWO_OPLOCK, BREAK_TO_TWO, BREAK_TO_NONE, BREAK_TO_TWO_TO_NONE, BREAK_TO_READ_CACHING, BREAK_TO_WRITE_CACHING, BREAK_TO_HANDLE_CACHING, or BREAK_TO_NO_CACHING:

// There could be a READ_CACHING-only oplock here. Those are broken later on.

If **Oplock.State** contains READ_CACHING, go to the *LeaveBreakToNone* label.

Set BREAK_TO_NONE in **Oplock.State**.

Notify the server of an oplock break according to the algorithm in section 3.1.5.17.3, setting the algorithm's parameters as follows:

BreakingOplockOpen equal to **Oplock.ExclusiveOpen**.

NewOplockLevel equal to LEVEL_NONE.

AcknowledgeRequired equal to TRUE.

OplockCompletionStatus equal to STATUS_SUCCESS.

(The operation does not end at this point; this call to 3.1.5.17.3 completes some earlier call to 3.1.5.17.1.)

Else If **Oplock.State** equals LEVEL_TWO_OPLOCK or (LEVEL_TWO_OPLOCK|READ_CACHING):

For each **Open** *ThisOpen* in **Oplock.IIOplocks**:

Remove *ThisOpen* from **Oplock.IIOplocks**.

Notify the server of an oplock break according to the algorithm in section 3.1.5.17.3, setting the algorithm's parameters as follows:

BreakingOplockOpen equal to *ThisOpen*.

NewOplockLevel equal to LEVEL_NONE.

AcknowledgeRequired equal to FALSE.

OplockCompletionStatus equal to STATUS_SUCCESS.

(The operation does not end at this point; this call to 3.1.5.17.3 completes some earlier call to 3.1.5.17.2.)

EndFor

If **Oplock.State** equals (LEVEL_TWO_OPLOCK|READ_CACHING):

Set **Oplock.State** equal to READ_CACHING.

Else

Set **Oplock.State** equal to NO_OPLOCK.

EndIf

Go to the *LeaveBreakToNone* label.

Else If **Oplock.State** contains BREAK_TO_TWO:

Clear BREAK_TO_TWO from **Oplock.State**.

Set BREAK_TO_TWO_TO_NONE in **Oplock.State**.

EndIf

If **Oplock.ExclusiveOpen** is not empty, and **Oplock.ExclusiveOpen.TargetOplockKey** equals **Open.TargetOplockKey**, go to the *LeaveBreakToNone* label.

The operation that called this algorithm MUST be made cancelable by inserting it into **CancelableOperations.CancelableOperationList**.

The operation that called this algorithm waits until the oplock break is acknowledged, as specified in section 3.1.5.18, or the operation is canceled.

EndIf

EndIf

EndIf

LeaveBreakToNone (goto destination label):

If *BreakCacheLevel* is not 0:

If **Oplock.State** contains any flags that are in *BreakCacheLevel*:

If **Oplock.ExclusiveOpen** is not empty, call the algorithm in section 3.1.4.12.2, passing **Open** as the **OperationOpen** parameter, **Oplock.ExclusiveOpen** as the **OplockOpen** parameter, and **Flags** as the **Flags** parameter. If the algorithm returns TRUE:

The algorithm returns at this point.

Switch (**Oplock.State**):

Case (READ_CACHING|HANDLE_CACHING|MIXED_R_AND_RH):

Case READ_CACHING:

Case (LEVEL_TWO_OPLOCK|READ_CACHING):

If *BreakCacheLevel* contains READ_CACHING:

For each **Open** *ThisOpen* in **Oplock.ROplocks**:

Call the algorithm in section 3.1.4.12.2, passing **Open** as the **OperationOpen** parameter, *ThisOpen* as the **OplockOpen** parameter, and **Flags** as the **Flags** parameter. If the algorithm returns FALSE:

Remove *ThisOpen* from **Oplock.ROplocks**.

Notify the server of an oplock break according to the algorithm in section 3.1.5.17.3, setting the algorithm's parameters as follows:

BreakingOplockOpen equal to *ThisOpen*.

NewOplockLevel equal to LEVEL_NONE.

AcknowledgeRequired equal to FALSE.

OplockCompletionStatus equal to STATUS_SUCCESS.

(The operation does not end at this point; this call to 3.1.5.17.3 completes some earlier call to 3.1.5.17.2.)

EndIf

EndFor

EndIf

If **Oplock.State** equals (READ_CACHING|HANDLE_CACHING|MIXED_R_AND_RH):

// Do nothing; FALL THROUGH to next Case statement.

Else

Recompute **Oplock.State** according to the algorithm in section 3.1.4.13, passing **Oplock** as the **ThisOplock** parameter.

EndCase

EndIf

Case (READ_CACHING|HANDLE_CACHING):

If *BreakCacheLevel* equals HANDLE_CACHING:

For each **Open** *ThisOpen* in **Oplock.RHOplocks**:

If *ThisOpen*.**OplockKey** does not equal **Open.OplockKey**:

Remove *ThisOpen* from **Oplock.RHOplocks**.

Notify the server of an oplock break according to the algorithm in section 3.1.5.17.3, setting the algorithm's parameters as follows:

BreakingOplockOpen equal to *ThisOpen*.

NewOplockLevel equal to READ_CACHING.

AcknowledgeRequired equal to TRUE.

OplockCompletionStatus equal to STATUS_SUCCESS.

(The operation does not end at this point; this call to 3.1.5.17.3 completes some earlier call to 3.1.5.17.2.)

Initialize a new **RHOpContext** object, setting its fields as follows:

RHOpContext.Open set to *ThisOpen*.

RHOpContext.BreakingToRead to TRUE.

Add the new **RHOpContext** object to **Oplock.RHBreakQueue**.

Set *NeedToWait* to TRUE.

EndIf

EndFor

Else If *BreakCacheLevel* contains both READ_CACHING and WRITE_CACHING:

For each **RHOpContext** *ThisContext* in **Oplock.RHBreakQueue**:

Call the algorithm in section 3.1.4.12.2, passing **Open** as the **OperationOpen** parameter, *ThisContext*.**Open** as the **OplockOpen** parameter, and **Flags** as the **Flags** parameter. If the algorithm returns FALSE:

Set *ThisContext*.**BreakingToRead** to FALSE.

If *BreakCacheLevel* contains HANDLE_CACHING:

Set *NeedToWait* to TRUE.

EndIf

EndIf

EndFor

For each **Open** *ThisOpen* in **Oplock.RHOplocks**:

Call the algorithm in section 3.1.4.12.2, passing **Open** as the **OperationOpen** parameter, *ThisOpen* as the **OplockOpen** parameter, and **Flags** as the **Flags** parameter. If the algorithm returns FALSE:

Remove *ThisOpen* from **Oplock.RHOplocks**.

Notify the server of an oplock break according to the algorithm in section 3.1.5.17.3, setting the algorithm's parameters as follows:

BreakingOplockOpen equal to *ThisOpen*.

NewOplockLevel equal to LEVEL_NONE.

AcknowledgeRequired equal to TRUE.

OplockCompletionStatus equal to STATUS_SUCCESS.

(The operation does not end at this point; this call to 3.1.5.17.3 completes some earlier call to 3.1.5.17.2.)

Initialize a new **RHOpContext** object, setting its fields as follows:

RHOpContext.Open set to *ThisOpen*.

RHOpContext.BreakingToRead to FALSE.

Add the new **RHOpContext** object to **Oplock.RHBreakQueue**.

If *BreakCacheLevel* contains HANDLE_CACHING:

Set *NeedToWait* to TRUE.

EndIf

EndIf

EndFor

EndIf

// If the oplock is explicitly losing HANDLE_CACHING, **RHBreakQueue** is not empty,

// and the algorithm has not yet decided to wait, this operation may have to wait if

// there is an oplock on **RHBreakQueue** with a non-matching key. This is done

// because even if this operation didn't cause a break of a currently-granted Read-

// Handle caching oplock, it may have done so had a currently-breaking oplock still

// been granted.

If (*NeedToWait* is FALSE) and

(**Oplock.RHBreakQueue** is empty) and

(*BreakCacheLevel* contains HANDLE_CACHING):

For each **RHOpContext** *ThisContex* in **Oplock.RHBreakQueue**:

If *ThisContext*.**Open.OplockKey** does not equal **Open.OplockKey**:

Set *NeedToWait* to TRUE.

Break out of the For loop.

EndIf

EndFor

EndIf

Recompute **Oplock.State** according to the algorithm in section 3.1.4.13, passing **Oplock** as the **ThisOplock** parameter.

EndCase

Case (READ_CACHING|HANDLE_CACHING|BREAK_TO_READ_CACHING):

If *BreakCacheLevel* contains READ_CACHING:

For each **RHOpContext** *ThisContext* in **Oplock.RHBreakQueue**:

Call the algorithm in section 3.1.4.12.2, passing **Open** as the **OperationOpen** parameter, *ThisContext*.**Open** as the **OplockOpen** parameter, and **Flags** as the **Flags** parameter. If the algorithm returns FALSE:

Set *ThisContext*.**BreakingToRead** to FALSE.

EndIf

Recompute **Oplock.State** according to the algorithm in section 3.1.4.13, passing **Oplock** as the **ThisOplock** parameter.

EndFor

EndIf

If *BreakCacheLevel* contains HANDLE_CACHING:

For each **RHOpContext** *ThisContext* in **Oplock.RHBreakQueue**:

If *ThisContext*.**Open.OplockKey** does not equal **Open.OplockKey**:

Set *NeedToWait* to TRUE.

Break out of the For loop.

EndIf

EndFor

EndIf

EndCase

Case (READ_CACHING|HANDLE_CACHING|BREAK_TO_NO_CACHING):

If *BreakCacheLevel* contains HANDLE_CACHING:

For each **RHOpContext** *ThisContext* in **Oplock.RHBreakQueue**:

If *ThisContext*.**Open.OplockKey** does not equal **Open.OplockKey**:

Set *NeedToWait* to TRUE.

Break out of the For loop.

EndIf

EndFor

EndIf

EndCase

Case (READ_CACHING|WRITE_CACHING|EXCLUSIVE):

If *BreakCacheLevel* contains both READ_CACHING and WRITE_CACHING:

Notify the server of an oplock break according to the algorithm in section 3.1.5.17.3, setting the algorithm's parameters as follows:

BreakingOplockOpen equal to **Oplock.ExclusiveOpen**.

NewOplockLevel equal to LEVEL_NONE.

AcknowledgeRequired equal to TRUE.

OplockCompletionStatus equal to STATUS_SUCCESS.

(The operation does not end at this point; this call to 3.1.5.17.3 completes some earlier call to 3.1.5.17.1.)

Set **Oplock.State** to (READ_CACHING|WRITE_CACHING|EXCLUSIVE|BREAK_TO_NO_CACHING).

Set *NeedToWait* to TRUE.

Else If *BreakCacheLevel* contains WRITE_CACHING:

Notify the server of an oplock break according to the algorithm in section 3.1.5.17.3, setting the algorithm's parameters as follows:

BreakingOplockOpen equal to **Oplock.ExclusiveOpen**.

NewOplockLevel equal to READ_CACHING.

AcknowledgeRequired equal to TRUE.

OplockCompletionStatus equal to STATUS_SUCCESS.

(The operation does not end at this point; this call to 3.1.5.17.3 completes some earlier call to 3.1.5.17.1.)

Set **Oplock.State** to (READ_CACHING|WRITE_CACHING|EXCLUSIVE|BREAK_TO_READ_CACHING).

Set *NeedToWait* to TRUE.

EndIf

EndCase

Case (READ_CACHING|WRITE_CACHING|HANDLE_CACHING|EXCLUSIVE):

If *BreakCacheLevel* equals WRITE_CACHING:

Notify the server of an oplock break according to the algorithm in section 3.1.5.17.3, setting the algorithm's parameters as follows:

BreakingOplockOpen equal to **Oplock.ExclusiveOpen**.

NewOplockLevel equal to (READ_CACHING|HANDLE_CACHING).

AcknowledgeRequired equal to TRUE.

OplockCompletionStatus equal to STATUS_SUCCESS.

(The operation does not end at this point; this call to 3.1.5.17.3 completes some earlier call to 3.1.5.17.1.)

Set **Oplock.State** to (READ_CACHING|WRITE_CACHING|HANDLE_CACHING|EXCLUSIVE|BREAK_TO_READ_CACHING|BREAK_TO_HANDLE_CACHING).

Set *NeedToWait* to TRUE.

Else If *BreakCacheLevel* equals HANDLE_CACHING:

Notify the server of an oplock break according to the algorithm in section 3.1.5.17.3, setting the algorithm's parameters as follows:

BreakingOplockOpen equal to **Oplock.ExclusiveOpen**.

NewOplockLevel equal to (READ_CACHING|WRITE_CACHING).

AcknowledgeRequired equal to TRUE.

OplockCompletionStatus equal to STATUS_SUCCESS.

(The operation does not end at this point; this call to 3.1.5.17.3 completes some earlier call to 3.1.5.17.1.)

Set **Oplock.State** to (READ_CACHING|WRITE_CACHING|HANDLE_CACHING|EXCLUSIVE|BREAK_TO_READ_CACHING|BREAK_TO_WRITE_CACHING).

Set *NeedToWait* to TRUE.

Else If *BreakCacheLevel* contains both READ_CACHING and WRITE_CACHING:

Notify the server of an oplock break according to the algorithm in section 3.1.5.17.3, setting the algorithm's parameters as follows:

BreakingOplockOpen equal to **Oplock.ExclusiveOpen**.

NewOplockLevel equal to LEVEL_NONE.

AcknowledgeRequired equal to TRUE.

OplockCompletionStatus equal to STATUS_SUCCESS.

(The operation does not end at this point; this call to 3.1.5.17.3 completes some earlier call to 3.1.5.17.1.)

Set **Oplock.State** to (READ_CACHING|WRITE_CACHING|HANDLE_CACHING|EXCLUSIVE|BREAK_TO_NO_CACHING).

Set *NeedToWait* to TRUE.

EndIf

EndCase

Case (READ_CACHING|WRITE_CACHING|EXCLUSIVE|BREAK_TO_READ_CACHING):

If *BreakCacheLevel* contains READ_CACHING:

Set **Oplock.State** to (READ_CACHING|WRITE_CACHING|EXCLUSIVE|BREAK_TO_NO_CACHING).

EndIf

If *BreakCacheLevel* contains either READ_CACHING or WRITE_CACHING:

Set *NeedToWait* to TRUE.

EndIf

EndCase

Case (READ_CACHING|WRITE_CACHING|EXCLUSIVE|BREAK_TO_NO_CACHING):

If *BreakCacheLevel* contains either READ_CACHING or WRITE_CACHING:

Set *NeedToWait* to TRUE.

EndIf

EndCase

Case (READ_CACHING|WRITE_CACHING|HANDLE_CACHING|EXCLUSIVE|BREAK_TO_READ_CACHING|BREAK_TO_WRITE_CACHING):

If *BreakCacheLevel* == WRITE_CACHING:

Set **Oplock.State** to (READ_CACHING|WRITE_CACHING|HANDLE_CACHING|EXCLUSIVE|BREAK_TO_READ_CACHING).

Else If *BreakCacheLevel* contains both READ_CACHING and WRITE_CACHING:

Set **Oplock.State** to (READ_CACHING|WRITE_CACHING|HANDLE_CACHING|EXCLUSIVE|BREAK_TO_NO_CACHING).

EndIf

Set *NeedToWait* to TRUE.

EndCase

Case (READ_CACHING|WRITE_CACHING|HANDLE_CACHING|EXCLUSIVE|BREAK_TO_READ_CACHING|BREAK_TO_HANDLE_CACHING):

If *BreakCacheLevel* == HANDLE_CACHING:

Set **Oplock.State** to (READ_CACHING|WRITE_CACHING|HANDLE_CACHING|EXCLUSIVE|BREAK_TO_READ_CACHING).

Else If *BreakCacheLevel* contains READ_CACHING:

Set **Oplock.State** to (READ_CACHING|WRITE_CACHING|HANDLE_CACHING|EXCLUSIVE|BREAK_TO_NO_CACHING).

EndIf

Set *NeedToWait* to TRUE.

EndCase

Case (READ_CACHING|WRITE_CACHING|HANDLE_CACHING|EXCLUSIVE|BREAK_TO_READ_CACHING):

If *BreakCacheLevel* contains READ_CACHING, set **Oplock.State** to (READ_CACHING|WRITE_CACHING|HANDLE_CACHING|EXCLUSIVE|BREAK_TO_NO_CACHING).

Set *NeedToWait* to TRUE.

EndCase

Case (READ_CACHING|WRITE_CACHING|HANDLE_CACHING|EXCLUSIVE|BREAK_TO_NO_CACHING):

Set *NeedToWait* to TRUE.

EndCase

EndSwitch

If *NeedToWait* is TRUE:

The operation that called this algorithm MUST be made cancelable by inserting it into **CancelableOperations.CancelableOperationList**.

The operation that called this algorithm waits until the oplock break is acknowledged, as specified in section 3.1.5.18, or the operation is canceled.

EndIf

EndIf

EndIf

EndIf

3.1.4.12.1 Algorithm for Request Processing After an Oplock Breaks

The inputs for this algorithm are:

OpenToRelease: The **Open** used in the request that caused the oplock to break

Pseudocode for the algorithm is as follows:

The request corresponding to **OpenToRelease** MUST resume from the point where it broke the oplock (that is, called section 3.1.4.12).

3.1.4.12.2 Algorithm to Compare Oplock Keys

Note: All of the information in this section is subject to change because it applies to a preliminary implementation of the protocol or structure.

The inputs for this algorithm are:

OperationOpen: The **Open** used in the request that may cause an oplock to break.

OplockOpen: The **Open** originally used to request the oplock, per section 3.1.5.17.

Flags: If unspecified it is considered to contain 0. Valid nonzero values are:

PARENT_OBJECT

This algorithm returns TRUE if the appropriate oplock key field of **OperationOpen** equals **OplockOpen.TargetOplockKey**, and FALSE otherwise.

Pseudocode for the algorithm is as follows:

If **OperationOpen** equals **OplockOpen**:

Return TRUE.

If both **OperationOpen.TargetOplockKey** and **OperationOpen.ParentOplockKey** are empty or both **OplockOpen.TargetOplockKey** and **OplockKey.ParentOplockKey** are empty:

Return FALSE.

If **OplockOpen.TargetOplockKey** is empty or

(**Flags** does not contain PARENT_OBJECT and **OperationOpen.TargetOplockKey** is empty):

Return FALSE.

If **Flags** contains PARENT_OBJECT and

OperationOpen.ParentOplockKey is empty:

Return FALSE.

If **Flags** contains PARENT_OBJECT:

If **OperationOpen.ParentOplockKey** equals **OplockOpen.TargetOplockKey**:

Return TRUE.

Else:

Return FALSE.

EndIf

Else:

If **OperationOpen.TargetOplockKey** equals **OplockOpen.TargetOplockKey**:

Return TRUE.

Else:

Return FALSE.

EndIf

EndIf

3.1.4.13 Algorithm to Recompute the State of a Shared Oplock

The inputs for this algorithm are:

ThisOplock: The **Oplock** on whose state is being recomputed.

Pseudocode for the algorithm is as follows:

If **ThisOplock.IIOplocks**, **ThisOplock.ROplocks**, **ThisOplock.RHOplocks**, and **ThisOplock.RHBreakQueue** are all empty:

Set **ThisOplock.State** to NO_OPLOCK.

Else If **ThisOplock.ROplocks** is not empty and either **ThisOplock.RHOplocks** or **ThisOplock.RHBreakQueue** are not empty:

Set **ThisOplock.State** to (READ_CACHING|HANDLE_CACHING|MIXED_R_AND_RH).

Else If **ThisOplock.ROplocks** is empty and **ThisOplock.RHOplocks** is not empty:

Set **ThisOplock.State** to (READ_CACHING|HANDLE_CACHING).

Else If **ThisOplock.ROplocks** is not empty and **ThisOplock.IIOplocks** is not empty:

Set **ThisOplock.State** to (READ_CACHING|LEVEL_TWO_OPLOCK).

Else If **ThisOplock.ROplocks** is not empty and **ThisOplock.IIOplocks** is empty:

Set **ThisOplock.State** to READ_CACHING.

Else If **ThisOplock.ROplocks** is empty and **ThisOplock.IIOplocks** is not empty:

Set **ThisOplock.State** to LEVEL_TWO_OPLOCK.

Else

// **ThisOplock.RHBreakQueue** MUST be non-empty by this point.

If **RHOpContext.BreakingToRead** is TRUE for every **RHOpContext** on **ThisOplock.RHBreakQueue**:

Set **ThisOplock.State** to (READ_CACHING|HANDLE_CACHING|BREAK_TO_READ_CACHING).

Else If **RHOpContext.BreakingToRead** is FALSE for every **RHOpContext** on **ThisOplock.RHBreakQueue**:

Set **ThisOplock.State** to (READ_CACHING|HANDLE_CACHING|BREAK_TO_NO_CACHING).

Else:

Set **ThisOplock.State** to (READ_CACHING|HANDLE_CACHING).

EndIf

EndIf

3.1.4.14 AccessCheck -- Algorithm to Perform a General Access Check

The inputs for this algorithm are:

SecurityContext: The **SecurityContext** of the user requesting access.

SecurityDescriptor: The security descriptor of the object to which access is requested, in the format specified in [MS-DTYP] section 2.4.6.

DesiredAccess: An ACCESS_MASK indicating type of access requested, as specified in [MS-DTYP] section 2.4.3.

This algorithm returns a Boolean value:

TRUE if the user has the necessary access to the object.

FALSE otherwise.

Pseudocode for the algorithm is as follows:

The object store MUST build a new *Token* object, in the format specified in [MS-DTYP] section 2.5.2, with fields initialized as follows:

SIDs set to **SecurityContext.SIDs**.

OwnerIndex set to **SecurityContext.OwnerIndex**.

PrimaryGroup set to **SecurityContext.PrimaryGroup**.

DefaultDACL set to **SecurityContext.DefaultDACL**.

SystemACLAccess set to TRUE if **SecurityContext.PrivilegeSet** contains "SeSecurityPrivilege", FALSE otherwise.

TakeOwnership set to TRUE if **SecurityContext.PrivilegeSet** contains "SeTakeOwnershipPrivilege", FALSE otherwise.

The object store MUST use the access check algorithm described in [MS-DTYP] section 2.5.3.2, with input values as follows:

SecurityDescriptor set to the **SecurityDescriptor** above.

Token set to *Token*.

Access Request mask set to **DesiredAccess**.

Object Tree set to NULL.

PrincipalSelfSubst set to NULL.

If the access check returns success, return TRUE; otherwise return FALSE.

3.1.4.15 BuildRelativeName -- Algorithm for Building the Relative Path Name for a Link

The inputs for this algorithm are:

Link: A **Link** whose relative path name we are building.

RootDirectory: A **DirectoryFile** indicating how far to walk up the directory hierarchy when building the relative path name.

This algorithm returns a Unicode string representing the portion of a Link's path name from **RootDirectory** to **Link** itself, inclusive. The returned string starts with a backslash and uses backslashes as path separators. If **Link** is not a descendant of **RootDirectory**, the algorithm returns an empty string to indicate this error.

Pseudocode for the algorithm is as follows:

If **Link.File** equals **RootDirectory**:

Return "\".

Else If **Link.File** equals **Link.File.Volume.RootDirectory**:

Return an empty string.

Else If **Link.ParentFile** equals **RootDirectory**:

Return "\" + **Link.Name**.

Else

Set *ParentRelativeName* to ***BuildRelativeName***(**Link.ParentFile**, **RootDirectory**).

If *ParentRelativeName* is empty:

Return an empty string.

Else

Return *ParentRelativeName* + "\" + **Link.Name**.

EndIf

EndIf

3.1.4.16 FindAllFiles: Algorithm for Finding All Files Under a Directory

The inputs for this algorithm are:

RootDirectory: A **DirectoryFile** ADM element indicating the top-level directory for the search.

This algorithm returns a list of files that are descendants of **RootDirectory**, including **RootDirectory** itself.

The algorithm uses the following local variables:

Lists of Files (initialized to empty): *FoundFiles*, *FilesToMerge*

Pseudocode for the algorithm follows:

```
Insert RootDirectory into FoundFiles.
For each Link in RootDirectory.DirectoryList:
    If Link.File.FileType is DirectoryFile:
        Set FilesToMerge to FindAllFiles(Link.File).
    Else:
        Set FilesToMerge to a list containing the single entry Link.File.
    EndIf
    For each File in FilesToMerge:
        If File is not an element of FoundFiles, insert File into FoundFiles.
    EndFor
EndFor
Return FoundFiles.
```

3.1.4.17 Algorithm for Noting That a File Has Been Modified

The inputs for this algorithm are as follows:

Open: The **Open** through which the file was modified.

The pseudocode for the algorithm is as follows:

```
If Open.UserSetModificationTime is FALSE, set Open.File.LastModificationTime to the current system time.
If Open.UserSetChangeTime is FALSE, set Open.File.LastChangeTime to the current system time.
If Open.UserSetAccessTime is FALSE, set Open.File.LastAccessTime to the current system time.
Set Open.File.FileAttributes.FILE_ATTRIBUTE_ARCHIVE to TRUE.
```

3.1.5 Higher-Layer Triggered Events

This section describes operations the object store performs in response to events triggered by higher-layer applications. The higher-layer application for this document is generally a server application that is processing requests for a local or remote client.

In performing these operations, the object store MAY make persistent changes to objects described in the abstract data model, section 3.1.1. If any operation fails, the object store SHOULD undo any persistent changes that were made prior to the failure, unless specifically noted otherwise in the operation.

In addition to the parameters explicitly listed, each operation in this section takes an implementation-specific parameter (**IORequest**) that uniquely identifies the in-progress I/O operation. The caller generates the **IORequest** value and passes it in as an additional parameter to the event. The **IORequest** parameter is used to support operation cancellation, as specified in section 3.1.5.19.

When an operation completes or is canceled the object store MUST remove the associated **IORequest** operation from **CancelableOperations.CancelableOperationList.**

3.1.5.1 Server Requests an Open of a File

Note: Some of the information in this section is subject to change because it applies to a preliminary implementation of the protocol or structure. For information about specific differences between versions, see the behavior notes that are provided in the Product Behavior appendix.

The server provides:

RootOpen: An **Open** to the root of the share.

PathName: A Unicode path relative to **RootOpen** for the file to be opened in the format specified in [MS-FSCC] section 2.1.5.

SecurityContext: The **SecurityContext** of the user performing the open.

DesiredAccess: A bitmask indicating requested access for the open, as specified in [MS-SMB2] section 2.2.13.1.

ShareAccess: A bitmask indicating sharing access for the open, as specified in [MS-SMB2] section 2.2.13.

CreateOptions: A bitmask of options for the open, as specified in [MS-SMB2] section 2.2.13.

CreateDisposition: The requested disposition for the open, as specified in [MS-SMB2] section 2.2.13.

DesiredFileAttributes: A bitmask of requested file attributes for the open, as specified in [MS-SMB2] section 2.2.13.

IsCaseInsensitive: A Boolean value. TRUE indicates that string comparisons performed in the context of this Open should be case-insensitive, otherwise they should be case-sensitive.

TargetOplockKey: A GUID value. This value could be empty.

UserCertificate: An ENCRYPTION_CERTIFICATE structure as specified in [MS-EFSR] section 2.2.8 and used when opening an encrypted stream. This value could be empty.

On completion, the object store MUST return:

Status: An NTSTATUS code that specifies the result.

On success it MUST also return:

CreateAction: A code defining the action taken by the open operation, as specified in [MS-SMB2] section 2.2.14 for the **CreateAction** field.

Open: The newly created **Open**.

On STATUS_REPARSE or STATUS_STOPPED_ON_SYMLINK it MUST also return:

ReparseData: The reparse point data associated with an existing file, in the format described in [MS-FSCC] section 2.1.2. The application MAY retry the open operation with a different **PathName** parameter constructed using **ReparseData**.

Pseudocode for the operation is as follows:

Phase 1 -- Parameter Validation:

The operation MUST be failed with STATUS_INVALID_PARAMETER under any of the following conditions:

If **RootOpen.File.FileType** is DataFile.

If **ShareAccess**, **CreateOptions**, **CreateDisposition**, or **FileAttributes** are not valid values for a file object as specified in [MS-SMB2] section 2.2.13.

If **CreateOptions.FILE_DIRECTORY_FILE** && **CreateOptions.FILE_NON_DIRECTORY_FILE**.

If (**CreateOptions.FILE_SYNCHRONOUS_IO_ALERT** || **Create.FILE_SYNCHRONOUS_IO_NONALERT**) && !**DesiredAccess.SYNCHRONIZE**.

If **CreateOptions.FILE_DELETE_ON_CLOSE** && !**DesiredAccess.DELETE**.

If **CreateOptions.FILE_SYNCHRONOUS_IO_ALERT** && **Create.FILE_SYNCHRONOUS_IO_NONALERT**.

If **CreateOptions.FILE_DIRECTORY_FILE** && (**CreateDisposition == SUPERSEDE** || **CreateDisposition == OVERWRITE** || **CreateDisposition == OVERWRITE_IF**).

If **CreateOptions.COMPLETE_IF_OPLOCKED** && **CreateOptions.FILE_RESERVE_OPFILTER**.

If **CreateOptions.FILE_NO_INTERMEDIATE_BUFFERING** && **DesiredAccess.FILE_APPEND_DATA**.

If **DesiredAccess** is zero or invalid (as specified in [MS-SMB2] section 2.2.13.1), the operation MUST be failed with STATUS_ACCESS_DENIED.

The operation MUST be failed with STATUS_OBJECT_NAME_INVALID under any of the following conditions:

If **PathName** is not valid as specified in [MS-FSCC] section 2.1.5.

If **PathName** contains a trailing backslash and **CreateOptions.FILE_NON_DIRECTORY_FILE** is TRUE.

If **DesiredFileAttributes.FILE_ATTRIBUTE_ENCRYPTED** is specified, then the object store MUST set **CreateOptions.FILE_NO_COMPRESSION**.

Phase 2 -- Volume State:

If **RootOpen.Volume.IsReadOnly** && (**CreateDisposition** == FILE_CREATE || **CreateDisposition** == FILE_SUPERSEDE || **CreateDisposition** == OVERWRITE || **CreateDisposition** == OVERWRITE_IF) then the operation MUST be failed with STATUS_MEDIA_WRITE_PROTECTED.

Phase 3 -- Initialization of **Open** Object:

The object store MUST build a new **Open** object with fields initialized as follows:

Open.RootOpen set to **RootOpen.**

Open.FileName formed by concatenating **RootOpen.FileName** + "\" + **FileName**, stripping any redundant backslashes and trailing backslashes.

Open.RemainingDesiredAccess set to **DesiredAccess**.

Open.SharingMode set to **ShareAccess**.

Open.Mode set to (**CreateOptions** & (FILE_WRITE_THROUGH | FILE_SEQUENTIAL_ONLY | FILE_NO_INTERMEDIATE_BUFFERING | FILE_SYNCHRONOUS_IO_ALERT | FILE_SYNCHRONOUS_IO_NONALERT | FILE_DELETE_ON_CLOSE)).

Open.IsCaseInsensitive set to **IsCaseInsensitive**.

Open.HasBackupAccess set to TRUE if **SecurityContext.PrivilegeSet** contains "SeBackupPrivilege".

Open.HasRestoreAccess set to TRUE if **SecurityContext.PrivilegeSet** contains "SeRestorePrivilege".

Open.HasCreateSymbolicLinkAccess set to TRUE if **SecurityContext.PrivilegeSet** contains "SeCreateSymbolicLinkPrivilege".

Open.HasManageVolumeAccess set to TRUE if **SecurityContext.PrivilegeSet** contains "SeManageVolumePrivilege".

Open.IsAdministrator set to TRUE if **SecurityContext.SIDs** contains the well-known SID BUILTIN_ADMINISTRATORS as defined in [MS-DTYP] section 2.4.2.4.

Open.TargetOplockKey set to **TargetOplockKey**.

Open.LastQuotaId set to -1.

All other fields set to zero.

Phase 4 -- Check for backup/restore intent

If **CreateOptions.FILE_OPEN_FOR_BACKUP_INTENT** is set and (**CreateDisposition** == FILE_OPEN || **CreateDisposition** == FILE_OPEN_IF || **CreateDisposition** ==

FILE_OVERWRITE_IF) and **Open.HasBackupAccess** is TRUE, then the object store SHOULD grant backup access as shown in the following pseudocode:

BackupAccess = (READ_CONTROL | ACCESS_SYSTEM_SECURITY | FILE_GENERIC_READ | FILE_TRAVERSE)

If **Open.RemainingDesiredAccess.MAXIMUM_ALLOWED** is set then:

Open.GrantedAccess |= *BackupAccess*

Else:

Open.GrantedAccess |= (**Open.RemainingDesiredAccess** & *BackupAccess*)

EndIf

Open.RemainingDesiredAccess &= ~**Open.GrantedAccess**

If **CreateOptions.FILE_OPEN_FOR_BACKUP_INTENT** is set and **Open.HasRestoreAccess** is TRUE, then the object store SHOULD grant restore access as shown in the following pseudocode:

RestoreAccess = (WRITE_DAC | WRITE_OWNER | ACCESS_SYSTEM_SECURITY | FILE_GENERIC_WRITE | FILE_ADD_FILE | FILE_ADD_SUBDIRECTORY | DELETE)

If **Open.RemainingDesiredAccess.MAXIMUM_ALLOWED** is set then:

Open.GrantedAccess |= *RestoreAccess*

Else:

Open.GrantedAccess |= (**Open.RemainingDesiredAccess** & *RestoreAccess*)

EndIf

Open.RemainingDesiredAccess &= ~**Open.GrantedAccess**

Phase 5 -- Parse pathname:

The object store MUST split **Open.FileName** into pathname components $PathName_1$ ... $PathName_n$, using the backslash ("\") character as a delimiter. The object store MUST further split each $PathName_i$ into a file name component $FileName_i$, stream name component $StreamName_i$, and stream type name component $StreamTypeName_i$, using the colon (":") character as a delimiter (FileNamei:StreamNamei:StreamTypeNamei). If StreamNamei or StreamTypeNamei is not present in the name, the value MUST be set to an empty string.

If any $StreamTypeName_i$ is "$INDEX_ALLOCATION" and the corresponding $StreamName_i$ has a value other than an empty string or "$I30", the operation SHOULD be failed with STATUS_INVALID_PARAMETER.

Phase 6 -- Location of file:

The object store MUST search for a filename matching **Open.FileName**. If **IsCaseInsensitive** is TRUE, then the search MUST be case-insensitive; otherwise it MUST be case-sensitive. Pseudocode for this search is as follows:

Set *ParentFile* = **RootOpen.File**.

// Examine each prefix pathname component in order.

For i = 1 to n-1: // n is the number of pathname components, from Phase 5.

Search *ParentFile*.**DirectoryList** for a **Link** where **Link.Name** or **Link.ShortName** matches $FileName_i$, If no such link is found, the operation MUST be failed with STATUS_OBJECT_PATH_NOT_FOUND.

If **Link.File.FileType** is not DirectoryFile, the operation MUST be failed with STATUS_NOT_A_DIRECTORY.

If **Open.GrantedAccess.FILE_TRAVERSE** is not set and ***AccessCheck***(**SecurityContext, Link.File.SecurityDescriptor**, FILE_TRAVERSE) returns FALSE, the operation MAY be failed with STATUS_ACCESS_DENIED.

If **Link.IsDeleted**, the operation MUST be failed with STATUS_DELETE_PENDING.

If **Link.File.IsSymbolicLink** is TRUE, the operation MUST be failed with **Status** set to STATUS_STOPPED_ON_SYMLINK and **ReparsePointData** set to **Link.File.ReparsePointData**.

Set *ParentFile* = **Link.File**.

EndFor

// Examine final pathname component.

Set *FileNameToOpen* to $FileName_n$, *StreamNameToOpen* to $StreamName_n$, and *StreamTypeNameToOpen* to $StreamTypeName_n$.

Search *ParentFile*.**DirectoryList** for a **Link** where **Link.Name** or **Link.ShortName** matches *FileNameToOpen*. If such a link is found:

Set **File** = **Link.File**.

Set **Open.File** to **File**.

Set **Open.Link** to **Link**.

Else:

If (**CreateDisposition ==** FILE_OPEN || **CreateDisposition ==** FILE_OVERWRITE), the operation MUST be failed with STATUS_OBJECT_NAME_NOT_FOUND.

If **RootOpen.Volume.IsReadOnly** then the operation MUST be failed with STATUS_MEDIA_WRITE_PROTECTED.

EndIf

Phase 7 -- Type of file to open:

The object store MUST use the following algorithm to determine which type of file is being opened:

If **CreateOptions.FILE_DIRECTORY_FILE** is TRUE then *FileTypeToOpen* = DirectoryFile.

Else if **CreateOptions.FILE_NON_DIRECTORY_FILE** is TRUE then *FileTypeToOpen* = DataFile.

Else if *StreamTypeNameToOpen* is "$INDEX_ALLOCATION" then *FileTypeToOpen* = DirectoryFile.

Else if *StreamTypeNameToOpen* is "$DATA" then *FileTypeToOpen* = DataFile.

Else if **Open.File** is not NULL and **Open.File.FileType** is DirectoryFile, then *FileTypeToOpen* = DirectoryFile.

Else if **PathName** contains a trailing backslash then *FileTypeToOpen* = DirectoryFile.

Else *FileTypeToOpen* = DataFile.

If *FileTypeToOpen* is DirectoryFile and **Open.File** is not NULL and **Open.File.FileType** is not DirectoryFile:

If **CreateDisposition** == FILE_CREATE then the operation MUST be failed with STATUS_OBJECT_NAME_COLLISION, else the operation MUST be failed with STATUS_NOT_A_DIRECTORY.

EndIf

If *FileTypeToOpen* is DataFile and *StreamNameToOpen* is empty and **Open.File** is not NULL and **Open.File.FileType** is DirectoryFile, the operation MUST be failed with STATUS_FILE_IS_A_DIRECTORY.

Phase 8 -- Completion of open

If **Open.File** is NULL, the object store MUST create a new file as described in section 3.1.5.1.1; otherwise the object store MUST open the existing file as described in section 3.1.5.1.2.

3.1.5.1.1 Creation of a New File

Note: Some of the information in this section is subject to change because it applies to a preliminary implementation of the protocol or structure. For information about specific differences between versions, see the behavior notes that are provided in the Product Behavior appendix.

Pseudocode for the operation is as follows:

If *FileTypeToOpen* is DirectoryFile and **DesiredFileAttributes.FILE_ATTRIBUTE_TEMPORARY** is set, the operation MUST be failed with STATUS_INVALID_PARAMETER.

If **DesiredFileAttributes.FILE_ATTRIBUTE_READONLY** and **CreateOptions.FILE_DELETE_ON_CLOSE** are both set, the operation MUST be failed with STATUS_CANNOT_DELETE.

If *StreamTypeNameToOpen* is non-empty and has a value other than "$DATA" or "$INDEX_ALLOCATION", the operation MUST be failed with STATUS_OBJECT_NAME_INVALID.

If **Open.RemainingDesiredAccess.ACCESS_SYSTEM_SECURITY** is set and **Open.GrantedAccess.ACCESS_SYSTEM_SECURITY** is not set and **SecurityContext.PrivilegeSet** does not contain "SeSecurityPrivilege", the operation MUST be failed with STATUS_ACCESS_DENIED.

If *FileTypeToOpen* is DataFile and **Open.GrantedAccess.FILE_ADD_FILE** is not set and ***AccessCheck***(**SecurityContext, Open.Link.ParentFile.SecurityDescriptor**, FILE_ADD_FILE) returns FALSE and **Open.HasRestoreAccess** is FALSE, the operation MUST be failed with STATUS_ACCESS_DENIED.

If *FileTypeToOpen* is DirectoryFile and **Open.GrantedAccess.FILE_ADD_SUBDIRECTORY** is not set and ***AccessCheck***(**SecurityContext, Open.Link.ParentFile.SecurityDescriptor**, FILE_ADD_SUBDIRECTORY) returns FALSE and **Open.HasRestoreAccess** is FALSE, the operation MUST be failed with STATUS_ACCESS_DENIED.

If the object store implements encryption and **DesiredFileAttributes.FILE_ATTRIBUTE_ENCRYPTED** is TRUE:

If **UserCertificate** is empty, the operation MUST be failed with **STATUS_CS_ENCRYPTION_NEW_ENCRYPTED_FILE**.

EndIf

The object store MUST build a new **File** object with fields initialized as follows:

File.FileType set to *FileTypeToOpen*.

File.FileID assigned a new value. The value chosen is implementation-specific but MUST be unique among all files present on **RootOpen.File.Volume**.<33>

File.FileNumber assigned a new value. The value chosen is implementation-specific but MUST be unique among all files present on **RootOpen.File.Volume**.<34>

File.FileAttributes set to **DesiredFileAttributes**.

File.CreationTime, **File.LastModificationTime**, **File.LastChangeTime**, and **File.LastAccessTime** all initialized to the current system time.

File.Volume set to **RootOpen.File.Volume**.

All other fields set to zero.

The object store MUST build a new **Link** object with fields initialized as follows:

Link.File set to **File**.

Link.ParentFile set to *ParentFile*.

All other fields set to zero.

If **File.FileType** is DataFile and **Open.IsCaseInsensitive** is TRUE, and tunnel caching is implemented, the object store MUST search **File.Volume.TunnelCacheList** for a *TunnelCacheEntry* where *TunnelCacheEntry***.ParentFile** equals **Link.ParentFile** and either (*TunnelCacheEntry***.KeyByShortName** is FALSE and *TunnelCacheEntry***.FileName** matches *FileNameToOpen*) or (*TunnelCacheEntry***.KeyByShortName** is TRUE and *TunnelCacheEntry***.FileShortName** matches *FileNameToOpen*). If such an entry is found, then:

Set **File.CreationTime** to *TunnelCacheEntry***.FileCreationTime**.

Set **File.ObjectId** to *TunnelCacheEntry***.FileObjectId**.

Set **Link.Name** to *TunnelCacheEntry***.FileName**.

Set **Link.ShortName** to *TunnelCacheEntry***.FileShortName** if that name is not already in use among all names and short names in **Link.ParentFile.DirectoryList**.

Remove *TunnelCacheEntry* from **File.Volume.TunnelCacheList**.

Else:

Set **Link.Name** to *FileNameToOpen*.

EndIf

If short names are enabled and **Link.ShortName** is empty, then the object store MUST create a short name as follows:

If **Link.Name** is 8.3-compliant as described in [MS-FSCC] section 2.1.5.2.1:

Set **Link.ShortName** to **Link.Name**.

Else:

Generate a new **Link.ShortName** that is 8.3-compliant as described in [MS-FSCC] section 2.1.5.2.1. The string chosen is implementation-specific, but MUST be unique among all names and short names present in **Link.ParentFile.DirectoryList**.

EndIf

EndIf

The object store MUST now grant the full requested access, as shown by the following pseudocode:

If **Open.RemainingDesiredAccess.MAXIMUM_ALLOWED** is set:

Open.GrantedAccess |= FILE_ALL_ACCESS

Else:

Open.GrantedAccess |= **Open.RemainingDesiredAccess**

EndIf

Open.RemainingDesiredAccess = 0

The object store MUST initialize **File.SecurityDescriptor.Dacl** to **SecurityContext.DefaultDACL**. The object store SHOULD append any inheritable security information from **Link.ParentFile.SecurityDescriptor** to **File.SecurityDescriptor**.

The object store MUST set **File.FileAttributes.FILE_ATTRIBUTE_NOT_CONTENT_INDEXED** to the value of **Link.ParentFile.FileAttributes.FILE_ATTRIBUTE_NOT_CONTENT_INDEXED**.

The object store MUST clear any attribute flags from **File.FileAttributes** that cannot be directly set by applications, as follows:

ValidSetAttributes = (FILE_ATTRIBUTE_READONLY | FILE_ATTRIBUTE_HIDDEN | FILE_ATTRIBUTE_SYSTEM | FILE_ATTRIBUTE_ARCHIVE | FILE_ATTRIBUTE_TEMPORARY | FILE_ATTRIBUTE_OFFLINE | FILE_ATTRIBUTE_NOT_CONTENT_INDEXED)

File.FileAttributes &= *ValidSetAttributes*

If **File.FileType** is DataFile, then the object store MUST set **File.FileAttributes.FILE_ATTRIBUTE_ARCHIVE**.

If **File.FileType** is DirectoryFile, then the object store MUST set **File.FileAttributes.FILE_ATTRIBUTE_DIRECTORY**.

If **Link.ParentFile.FileAttributes.FILE_ATTRIBUTE_ENCRYPTED** or **DesiredFileAttributes.FILE_ATTRIBUTE_ENCRYPTED** is set, then the object store MUST set **File.FileAttributes.FILE_ATTRIBUTE_ENCRYPTED**.

If **Link.ParentFile.FileAttributes.FILE_ATTRIBUTE_COMPRESSED** is set and **CreateOptions.FILE_NO_COMPRESSION** is not set, then the object store MUST set **File.FileAttributes.FILE_ATTRIBUTE_COMPRESSED**.

If **Link.ParentFile.FileAttributes.FILE_ATTRIBUTE_INTEGRITY_STREAM** is set or **DesiredFileAttributes.FILE_ATTRIBUTE_INTEGRITY_STREAM** is set, then the object store MUST set **File.FileAttributes.FILE_ATTRIBUTE_INTEGRITY_STREAM**.<35>

If **Link.ParentFile.FileAttributes.FILE_ATTRIBUTE_NO_SCRUB_DATA** is set or **DesiredFileAttributes.FILE_ATTRIBUTE_NO_SCRUB_DATA** is set, then the object store MUST set **File.FileAttributes.FILE_ATTRIBUTE_NO_SCRUB_DATA**.<36>

If the object store implements encryption and **File.FileAttributes.FILE_ATTRIBUTE_ENCRYPTED** is TRUE, insert **UserCertificate** into **File.UserCertificateList**.

If **File.FileType** is DataFile and *StreamNameToOpen* is not empty, then the object store MUST create a default unnamed stream for the file as follows:<37>

Build a new **Stream** object **DefaultStream** with all fields initially set to zero.

Set **DefaultStream.File** to **File**.

If the object store implements encryption and **File.FileAttributes.FILE_ATTRIBUTE_ENCRYPTED** is TRUE, set **DefaultStream.IsEncrypted** to TRUE.

Add **DefaultStream** to **File.StreamList**.

EndIf

If *StreamTypeNameToOpen* is empty or "$DATA", then the object store MUST create a new data stream for the file as follows:

Build a new **Stream** object with all fields initially set to zero.

Set **Stream.StreamType** to DataStream.

Set **Stream.Name** to *StreamNameToOpen*.

Set **Stream.File** to **File**.

Add **Stream** to **File.StreamList**.

Set **Open.Stream** to **Stream**.

Else the object store MUST create a new directory stream as follows:

Build a new **Stream** object with all fields initially set to zero.

Set **Stream.StreamType** to DirectoryStream.

Set **Stream.File** to **File**.

Add **Stream** to **File.StreamList**.

Set **Open.Stream** to **Stream**.

EndIf

If the object store implements encryption and **File.FileAttributes.FILE_ATTRIBUTE_ENCRYPTED** is TRUE:

If **File.FileType** is DataFile, set **Stream.IsEncrypted** to TRUE.

EndIf

The object store MUST set **Open.File** to **File**.

The object store MUST set **Open.Link** to **Link**.

The object store MUST insert **Link** into **File.LinkList**.

The object store MUST insert **Link** into **Link.ParentFile.DirectoryList**.

The object store MUST update **Link.ParentFile.LastModificationTime**, **Link.ParentFile.LastChangeTime**, and **Link.ParentFile.LastAccessTime** to the current system time.

If the **Oplock** member of the **DirectoryStream** in **Link.ParentFile.StreamList** (hereinafter referred to as *ParentOplock*) is not empty, the object store MUST check for an oplock break on the parent according to the algorithm in section 3.1.4.12, with input values as follows:

Open equal to this operation's **Open**

Oplock equal to *ParentOplock*

Operation equal to "OPEN"

Flags equal to "PARENT_OBJECT"

The object store MUST insert **File** into **File.Volume.OpenFileList**.

The object store MUST insert **Open** into **File.OpenList**.

If **File.FileType** is DirectoryFile:

FilterMatch = FILE_NOTIFY_CHANGE_DIR_NAME

Else:

FilterMatch = FILE_NOTIFY_CHANGE_FILE_NAME

EndIf

The object store MUST send directory change notification as per section 3.1.4.1 with **Volume** equal to **File.Volume**, **Action** equal to FILE_ACTION_ADDED, **FilterMatch** equal to *FilterMatch*, and **FileName** equal to **Open.FileName**.

The object store MUST return:

Status set to STATUS_SUCCESS.

CreateAction set to FILE_CREATED.

The **Open** object created previously.

3.1.5.1.2 Open of an Existing File

Note: Some of the information in this section is subject to change because it applies to a preliminary implementation of the protocol or structure. For information about specific differences between versions, see the behavior notes that are provided in the Product Behavior appendix.

Files that require knowledge of extended attributes cannot be opened by applications that do not understand extended attributes. If **CreateOptions.FILE_NO_EA_KNOWLEDGE** is set and (*FileTypeToOpen* is DirectoryFile or (*FileTypeToOpen* is DataFile and *StreamNameToOpen* is empty)) and **File.ExtendedAttributes** contains an *ExistingEa* where *ExistingEa***.Flags.FILE_NEED_EA** is set, the operation MUST be failed with STATUS_ACCESS_DENIED.

Pseudocode for the operation is as follows:

If **CreateOptions.FILE_OPEN_REPARSE_POINT** is not set and **File.ReparsePointTag** is not empty, then the operation MUST be failed with **Status** set to STATUS_REPARSE and **ReparsePointData** set to **File.ReparsePointData**.

If *FileTypeToOpen* is DirectoryFile:

If **CreateDisposition** is FILE_OPEN or FILE_OPEN_IF then:

Perform access checks as described in section 3.1.5.1.2.1. If this fails with STATUS_SHARING_VIOLATION:

If **Open.Stream.Oplock** is not empty and **Open.Stream.Oplock.State** contains HANDLE_CACHING, the object store MUST check for an oplock break according to the algorithm in section 3.1.4.12, with input values as follows:

Open equal to this operation's **Open**

Oplock equal to **Open.Stream.Oplock**

Operation equal to "OPEN_BREAK_H"

Perform access checks as described in section 3.1.5.1.2.1. If this fails, the request MUST be failed with the same status.

EndIf

Perform sharing access checks as described in section 3.1.5.1.2.2. If this fails with STATUS_SHARING_VIOLATION:

If **Open.Stream.Oplock** is not empty and **Open.Stream.Oplock.State** contains HANDLE_CACHING, the object store MUST check for an oplock break according to the algorithm in section 3.1.4.12, with input values as follows:

Open equal to this operation's **Open**

Oplock equal to **Open.Stream.Oplock**

Perform sharing access checks as described in section 3.1.5.1.2.2. If this fails, the request MUST be failed with the same status.

EndIf

Set **CreateAction** to FILE_OPENED.

Else:

// Existing directories cannot be overwritten/superseded.

If **File** == **File.Volume.RootDirectory**, then the operation MUST be failed with STATUS_ACCESS_DENIED, else the operation MUST be failed with STATUS_OBJECT_NAME_COLLISION.

EndIf

Else if *FileTypeToOpen* is DataFile:

The object store MUST search **File.StreamList** for a **Stream** with **Stream.Name** matching *StreamNameToOpen*. If **IsCaseInsensitive** is TRUE, then the search MUST be case-insensitive; otherwise it MUST be case-sensitive.

If **Stream** was found:

Set **Open.Stream** to **Stream**.

If **CreateDisposition** is FILE_CREATE, then the operation MUST be failed with STATUS_OBJECT_NAME_COLLISION.

If **CreateDisposition** is FILE_OPEN or FILE_OPEN_IF:

If **Open.Stream.Oplock** is not empty and **Open.Stream.Oplock.State** contains BATCH_OPLOCK, the object store MUST check for an oplock break according to the algorithm in section 3.1.4.12, with input values as follows:

Open equal to this operation's **Open**

Oplock equal to **Open.Stream.Oplock**

Operation equal to "OPEN"

OpParams containing two members:

DesiredAccess equal to this operation's **DesiredAccess**

CreateDisposition equal to this operation's **CreateDisposition**

Perform access checks as described in section 3.1.5.1.2.1. If this fails with STATUS_SHARING_VIOLATION:

If **Open.Stream.Oplock** is not empty and **Open.Stream.Oplock.State** contains HANDLE_CACHING, the object store MUST check for an oplock break according to the algorithm in section 3.1.4.12, with input values as follows:

Open equal to this operation's **Open**

Oplock equal to **Open.Stream.Oplock**

Operation equal to "OPEN_BREAK_H"

Perform access checks as described in section 3.1.5.1.2.1. If this fails, the request MUST be failed with the same status.

EndIf

Perform sharing access checks as described in section 3.1.5.1.2.2. If this fails with STATUS_SHARING_VIOLATION:

If **Open.Stream.Oplock** is not empty and **Open.Stream.Oplock.State** contains HANDLE_CACHING, the object store MUST check for an oplock break according to the algorithm in section 3.1.4.12, with input values as follows:

Open equal to this operation's **Open**

Oplock equal to **Open.Stream.Oplock**

Operation equal to "OPEN_BREAK_H"

Perform sharing access checks as described in section 3.1.5.1.2.2. If this fails, the request MUST be failed with the same status.

EndIf

Set **CreateAction** to FILE_OPENED.

Else:

If **File.Volume.IsReadOnly** is TRUE, the operation MUST be failed with STATUS_MEDIA_WRITE_PROTECTED.

If **Open.Stream.Oplock** is not empty and **Open.Stream.Oplock.State** contains BATCH_OPLOCK, the object store MUST check for an oplock break according to the algorithm in section 3.1.4.12, with input values as follows:

Open equal to this operation's **Open**

Oplock equal to **Open.Stream.Oplock**

Operation equal to "OPEN"

OpParams containing two members:

DesiredAccess equal to this operation's **DesiredAccess**

CreateDisposition equal to this operation's **CreateDisposition**

If **Stream.Name** is empty:

If **File.FileAttributes**.FILE_ATTRIBUTE_HIDDEN is TRUE and **DesiredFileAttributes**.FILE_ATTRIBUTE_HIDDEN is FALSE, then the operation MUST be failed with STATUS_ACCESS_DENIED.

If **File.FileAttributes**.FILE_ATTRIBUTE_SYSTEM is TRUE and **DesiredFileAttributes**.FILE_ATTRIBUTE_SYSTEM is FALSE, then the operation MUST be failed with STATUS_ACCESS_DENIED.

Set **DesiredFileAttributes**.FILE_ATTRIBUTE_ARCHIVE to TRUE.

Set **DesiredFileAttributes**.FILE_ATTRIBUTE_NORMAL to FALSE.

Set **DesiredFileAttributes**.FILE_ATTRIBUTE_NOT_CONTENT_INDEXED to FALSE.

If **File.FileAttributes**.FILE_ATTRIBUTE_ENCRYPTED is TRUE, then set **DesiredFileAttributes**.FILE_ATTRIBUTE_ENCRYPTED to TRUE.

If **Open.HasRestoreAccess** is TRUE, then the object store MUST set **Open.GrantedAccess**.FILE_WRITE_EA to TRUE. Otherwise, the object store MUST set **Open.RemainingDesiredAccess**.FILE_WRITE_EA to TRUE.

If **Open.HasRestoreAccess** is TRUE, then the object store MUST set **Open.GrantedAccess**.FILE_WRITE_ATTRIBUTES to TRUE. Otherwise, the object store MUST set **Open.RemainingDesiredAccess**.FILE_WRITE_ATTRIBUTES to TRUE.

EndIf

If **CreateDisposition** is FILE_SUPERSEDE:

If **Open.HasRestoreAccess** is TRUE, then the object store MUST set **Open.GrantedAccess**.DELETE to TRUE. Otherwise, the object store MUST set **Open.RemainingDesiredAccess**.DELETE to TRUE.

Else:

If **Open.HasRestoreAccess** is TRUE, then the object store MUST set **Open.GrantedAccess**.FILE_WRITE_DATA to TRUE. Otherwise, the object store MUST set **Open.RemainingDesiredAccess**.FILE_WRITE_DATA to TRUE.

EndIf

Open.RemainingDesiredAccess &= ~**Open.GrantedAccess**

Perform access checks as described in section 3.1.5.1.2.1. If this fails with STATUS_SHARING_VIOLATION:

If **Open.Stream.Oplock** is not empty and **Open.Stream.Oplock.State** contains HANDLE_CACHING, the object store MUST check for an oplock break according to the algorithm in section 3.1.4.12, with input values as follows:

Open equal to this operation's **Open**

Oplock equal to **Open.Stream.Oplock**

Operation equal to "OPEN_BREAK_H"

Perform access checks as described in section 3.1.5.1.2.1. If this fails, the request MUST be failed with the same status.

EndIf

Perform sharing access checks as described in section 3.1.5.1.2.2. If this fails with STATUS_SHARING_VIOLATION:

If **Open.Stream.Oplock** is not empty and **Open.Stream.Oplock.State** contains HANDLE_CACHING, the object store MUST check for an oplock break according to the algorithm in section 3.1.4.12, with input values as follows:

Open equal to this operation's **Open**

Oplock equal to **Open.Stream.Oplock**

Operation equal to "OPEN_BREAK_H"

Perform sharing access checks as described in section 3.1.5.1.2.2. If this fails, the request MUST be failed with the same status.

EndIf

Note that the file has been modified as specified in section 3.1.4.17 with **Open** equal to **Open**.

If **CreateDisposition** is FILE_SUPERSEDE, the object store MUST set **CreateAction** to FILE_SUPERSEDED; otherwise, it MUST set **CreateAction** to FILE_OVERWRITTEN.

EndIf

Else: // **Stream** not found.

If **CreateDisposition** is FILE_OPEN or FILE_OVERWRITE, the operation MUST be failed with STATUS_OBJECT_NAME_NOT_FOUND.

If **Open.GrantedAccess**.FILE_WRITE_DATA is not set and **Open.RemainingDesiredAccess**.FILE_WRITE_DATA is not set:

If **Open.HasRestoreAccess** is TRUE, then the object store MUST set **Open.GrantedAccess**.FILE_WRITE_DATA to TRUE; otherwise, the object store MUST set **Open.RemainingDesiredAccess**.FILE_WRITE_DATA to TRUE.

EndIf

Perform access checks as described in section 3.1.5.1.2.1. If this fails, the request MUST be failed with the same status.

If **File.Volume.IsReadOnly** is TRUE, the operation MUST be failed with STATUS_MEDIA_WRITE_PROTECTED.

Update **File.LastChangeTime** to the current time.

Set **File.FileAttributes**.FILE_ATTRIBUTE_ARCHIVE to TRUE.

Build a new **Stream** object with all fields initially set to zero.

Set **Stream.StreamType** to DataStream.

Set **Stream.Name** to *StreamNameToOpen*.

Set **Stream.File** to **File**.

Add **Stream** to **File.StreamList**.

Set **Open.Stream** to **Stream**.

Set **CreateAction** to FILE_CREATED.

EndIf.

EndIf

If the object store implements encryption:

If (**CreateAction** is FILE_OVERWRITTEN or FILE_SUPERSEDED) and (**Stream.Name** is empty) and (**DesiredFileAttributes**.FILE_ATTRIBUTE_ENCRYPTED is TRUE) and (**File.FileAttributes**.FILE_ATTRIBUTE_ENCRYPTED is FALSE), then:

If **File.OpenList** is non-empty, then the operation MUST be failed with STATUS_SHARING_VIOLATION.

If **CreateAction** is one of FILE_CREATED, FILE_OVERWRITTEN or FILE_SUPERSEDED, then:

The object store MUST set *FilterMatch* to a set of flags capturing modifications to the existing file's persistent attributes performed during the Open operation.

Send directory change notification as per section 3.1.4.1, with **Volume** equal to **File.Volume**, **Action** equal to FILE_ACTION_MODIFIED, **FilterMatch** equal to *FilterMatch*, and **FileName** equal to **Open.FileName**.

EndIf

If **CreateAction** is FILE_CREATED, then the object store MUST insert **Stream** into **File.StreamList**.

If **File** is not in **File.Volume.OpenFileList**, the object store MUST insert it.

The object store MUST insert **Open** into **File.OpenList**.

The object store MUST return:

Status set to STATUS_SUCCESS.

CreateAction set to FILE_OPENED.

The **Open** object created previously.

3.1.5.1.2.1 Algorithm to Check Access to an Existing File

The inputs to the algorithm are:

Open: The **Open** for an in-progress Open operation to an existing file.

On completion, the algorithm returns:

Status: An NTSTATUS code that specifies the result of the access check.

This object store MUST perform access checks when opening an existing file, making use of the file's security descriptor and possibly the parent file's security descriptor.

Pseudocode for these checks is as follows:

If **Open.File.FileType** is DataFile and (**File.FileAttributes**.FILE_ATTRIBUTE_READONLY && (**DesiredAccess**.FILE_WRITE_DATA || **DesiredAccess**.FILE_APPEND_DATA)), then return STATUS_ACCESS_DENIED.

If ((**File.FileAttributes**.FILE_ATTRIBUTE_READONLY || **File.Volume.IsReadOnly**) && **CreateOptions.**FILE_DELETE_ON_CLOSE), then return STATUS_CANNOT_DELETE.

If **Open.RemainingDesiredAccess** is nonzero:

If **Open.RemainingDesiredAccess**.MAXIMUM_ALLOWED is TRUE:

For each *Access* Flag in FILE_ALL_ACCESS, the object store MUST set **Open.GrantedAccess.***Access* if ***AccessCheck***(**SecurityContext, File.SecurityDescriptor**, *Access*) returns TRUE.

If **File.FileAttributes.**FILE_ATTRIBUTE_READONLY or **File.Volume.IsReadOnly**, then the object store MUST clear (FILE_WRITE_DATA | FILE_APPEND_DATA | FILE_ADD_SUBDIRECTORY | FILE_DELETE_CHILD) from **Open.GrantedAccess**.

Else:

For each *Access* Flag in **Open.RemainingDesired**.*Access*, the object store MUST set **Open.GrantedAccess**.*Access* if ***AccessCheck***(**SecurityContext, File.SecurityDescriptor**, *Access*) returns TRUE.

EndIf

If (**Open.RemainingDesiredAccess**.MAXIMUM_ALLOWED || **Open.RemainingDesiredAccess.**DELETE), the object store MUST set **Open.GrantedAccess.**DELETE if ***AccessCheck***(**SecurityContext, Open.Link.ParentFile.SecurityDescriptor**, FILE_DELETE_CHILD) returns TRUE.

If (**Open.RemainingDesiredAccess**.MAXIMUM_ALLOWED || **Open.RemainingDesiredAccess.**FILE_READ_ATTRIBUTES), the object store MUST set **Open.GrantedAccess.**FILE_READ_ATTRIBUTES if ***AccessCheck***(**SecurityContext, Open.Link.ParentFile.SecurityDescriptor**, FILE_LIST_DIRECTORY) returns TRUE.

Open.RemainingDesiredAccess &= ~(**Open.GrantedAccess** | MAXIMUM_ALLOWED)

If **Open.RemainingDesiredAccess** is nonzero, then return STATUS_ACCESS_DENIED.

EndIf

Since deletion of a file's primary stream implies deletion of the entire file, including any **alternate data streams**, the object store MUST check for sharing conflicts involving deletion of the primary stream and the sharing modes of all opens to the file.

Pseudocode for these checks is as follows:

If **Open.SharingMode**.FILE_SHARE_DELETE is FALSE and **Open.GrantedAccess** contains any one or more of (FILE_EXECUTE | FILE_READ_DATA | FILE_WRITE_DATA | FILE_APPEND_DATA):

For each *ExistingOpen* is **Open.File.OpenList**:

If *ExistingOpen*.**Mode**.FILE_DELETE_ON_CLOSE is TRUE and (*ExistingOpen*.**Stream.StreamType** is DirectoryStream or *ExistingOpen*.**Stream.Name** is empty), then return STATUS_SHARING_VIOLATION.

EndFor

EndIf

If **Open.GrantedAccess**.DELETE is TRUE and (**Open.Stream.StreamType** is DirectoryStream or **Open.Stream.Name** is empty):

For each *ExistingOpen* in **Open.File.OpenList**:

If *ExistingOpen*.**SharingMode**.FILE_SHARE_DELETE is FALSE, then return STATUS_SHARING_VIOLATION.

EndFor

EndIf

Return STATUS_SUCCESS.

3.1.5.1.2.2 Algorithm to Check Sharing Access to an Existing Stream or Directory

Note: Some of the information in this section is subject to change because it applies to a preliminary implementation of the protocol or structure. For information about specific differences between versions, see the behavior notes that are provided in the Product Behavior appendix.

The inputs to the algorithm are:

Open: The **Open** for an in-progress Open operation to an existing stream or directory.

On completion, the algorithm returns:

Status: An NTSTATUS code that specifies the result of the sharing check.

The object store MUST perform sharing checks when opening an existing stream or directory.

Pseudocode for these checks is as follows:

If ***AccessCheck***(**SecurityContext, Open.Link.ParentFile.SecurityDescriptor**, FILE_WRITE_DATA) returns FALSE, the object store MUST set **Open.SharingMode**.FILE_SHARE_READ to TRUE.

If **DesiredAccess** contains any of (FILE_READ_DATA | FILE_EXECUTE | FILE_WRITE_DATA | FILE_APPEND_DATA | DELETE):

For each *ExistingOpen* in **Open.File.OpenList**:

If *ExistingOpen*.**Stream** equals **Open.Stream** and *ExistingOpen*.**GrantedAccess** contains any of (FILE_READ_DATA | FILE_EXECUTE | FILE_WRITE_DATA | FILE_APPEND_DATA | DELETE), then return STATUS_SHARING_VIOLATION under any of the following conditions:

If *ExistingOpen*.**SharingMode**.FILE_SHARE_READ is FALSE and **DesiredAccess** contains either FILE_READ_DATA or FILE_EXECUTE

If *ExistingOpen*.**SharingMode**.FILE_SHARE_WRITE is FALSE and **DesiredAccess** contains either FILE_WRITE_DATA or FILE_APPEND_DATA

If *ExistingOpen*.**SharingMode**.FILE_SHARE_DELETE is FALSE and *ExistingOpen* contains DELETE

If **Open.SharingMode**.FILE_SHARE_READ is FALSE and *ExistingOpen*.**GrantedAccess** contains either FILE_READ_DATA or FILE_EXECUTE

If **Open.SharingMode**.FILE_SHARE_WRITE is FALSE and *ExistingOpen*.**GrantedAccess** contains either FILE_WRITE_DATA or FILE_APPEND_DATA

If **Open.SharingMode**.FILE_SHARE_READ is FALSE and *ExistingOpen*.**GrantedAccess** contains DELETE

EndIf

EndFor

EndIf

If **Open.Stream.Oplock** is not empty, the object store MUST check for an oplock break according to the algorithm in section 3.1.4.12, with input values as follows:

Open equal to this operation's **Open**

Oplock equal to **Open.Stream.Oplock**

Operation equal to "OPEN"

OpParams containing two members:

DesiredAccess equal to this operation's **DesiredAccess**

CreateDisposition equal to this operation's **CreateDisposition**

EndIf

Return STATUS_SUCCESS.

3.1.5.2 Server Requests a Read

Note: Some of the information in this section is subject to change because it applies to a preliminary implementation of the protocol or structure. For information about specific differences between versions, see the behavior notes that are provided in the Product Behavior appendix.

The server provides:

Open: The **Open** of the DataFile to read from.

ByteOffset: The absolute byte offset in the stream from which to read data.

ByteCount: The requested number of bytes to read.

On completion, the object store MUST return:

Status: An NTSTATUS code that specifies the result.

OutputBuffer: An array of bytes that were read.

BytesRead: The number of bytes that were read.

Pseudocode for the operation is as follows:

If **Open.Mode.FILE_NO_INTERMEDIATE_BUFFERING** is TRUE & (**ByteOffset** >= 0), the operation MUST be failed with STATUS_INVALID_PARAMETER under any of the following conditions:

(**ByteOffset** % **Open.File.Volume.LogicalBytesPerSector**) is not zero.

(**ByteCount** % **Open.File.Volume.LogicalBytesPerSector**) is not zero.

If **ByteOffset** is negative, then the operation MUST be failed with STATUS_INVALID_PARAMETER.

If (**ByteOffset** + **ByteCount**) is larger than MAXLONGLONG (0x7fffffffffffffff), the operation MUST be failed with STATUS_INVALID_PARAMETER.

If **ByteCount** is zero, the object store MUST return:

BytesRead set to zero.

Status set to STATUS_SUCCESS.

Set *RequestedByteCount* to **ByteCount**.

If **Open.Stream.Oplock** is not empty, the object store MUST check for an oplock break according to the algorithm in section 3.1.4.12, with input values as follows:

Open equal to this operation's **Open**

Oplock equal to **Open.Stream.Oplock**

Operation equal to "READ"

OpParams empty

Determine if the read is in conflict with an existing byte range lock on **Open.Stream** using the algorithm described in section 3.1.4.10 (with **ByteOffset** set to **ByteOffset**, **Length** set to **ByteCount**, **IsExclusive** set to FALSE, **LockIntent** set to FALSE and **Open** set to **Open**). If the algorithm returns TRUE, the operation MUST be failed with STATUS_FILE_LOCK_CONFLICT.

If **ByteOffset** >= **Open.Stream.Size**, the operation MUST be failed with STATUS_END_OF_FILE.

If (**ByteOffset** + **ByteCount**) >= **Open.Stream.Size**, truncate **ByteCount** to (**Open.Stream.Size** - **ByteOffset**) and then set *RequestedByteCount* to **ByteCount**.

If **Open.Mode.FILE_NO_INTERMEDIATE_BUFFERING** is TRUE:

The object store MUST write any unwritten cached data for this range of the stream to disk.

The object store MUST remove from the cache any cached data for this range of the stream.

If (**ByteOffset** >= **Open.Stream.ValidDataLength**):

If **Open.Mode.FILE_SYNCHRONOUS_IO_ALERT** is TRUE or **Open.Mode.FILE_SYNCHRONOUS_IO_NONALERT** is TRUE, the object store MUST set **Open.CurrentByteOffset** to (**ByteOffset** + **ByteCount**).

If **Open.File.UserSetAccessTime** is FALSE, the object store MUST update **Open.File.LastAccessTime** to the current system time.

The object store MUST return:

BytesRead set to **ByteCount**.

OutputBuffer filled with **ByteCount** zero(s).

Status set to STATUS_SUCCESS.

EndIf

If ((**ByteOffset** + **ByteCount**) >= **Open.Stream.ValidDataLength**), truncate **ByteCount** to (**Open.Stream.ValidDataLength** - **ByteOffset**).

Set *BytesToRead* to ***BlockAlign***(**ByteCount, Open.File.Volume.LogicalBytesPerSector**).

Read *BytesToRead* bytes from the disk at offset **ByteOffset** for this stream into **OutputBuffer**. If the read from the disk failed, the operation MUST be failed with the same error status**.**

If *RequestedByteCount* > **ByteCount**, zero out **OutputBuffer** between **ByteCount** and *RequestedByteCount*.

If **Open.Mode.FILE_SYNCHRONOUS_IO_ALERT** is TRUE or **Open.Mode.FILE_SYNCHRONOUS_IO_NONALERT** is TRUE, the object store MUST set **Open.CurrentByteOffset** to (**ByteOffset** + *RequestedByteCount*).

If **Open.File.UserSetAccessTime** is FALSE, the object store MUST update **Open.File.LastAccessTime** to the current system time.

Upon successful completion of the operation, the object store MUST return:

BytesRead set to *RequestedByteCount*.

Status set to STATUS_SUCCESS.

Else

Read **ByteCount** bytes at offset **ByteOffset** from the cache for this stream into **OutputBuffer**.

If **Open.Mode.FILE_SYNCHRONOUS_IO_ALERT** is TRUE or **Open.Mode.FILE_SYNCHRONOUS_IO_NONALERT** is TRUE, the object store MUST set **Open.CurrentByteOffset** to (**ByteOffset** + **ByteCount**).

If **Open.File.UserSetAccessTime** is FALSE, the object store MUST update **Open.File.LastAccessTime** to the current system time.

Upon successful completion of the operation, the object store MUST return:

BytesRead set to **ByteCount**.

Status set to STATUS_SUCCESS.

EndIf

3.1.5.3 Server Requests a Write

Note: Some of the information in this section is subject to change because it applies to a preliminary implementation of the protocol or structure. For information about specific differences between versions, see the behavior notes that are provided in the Product Behavior appendix.

The server provides:

Open: The **Open** of the DataFile to write to.

InputBuffer: An array of bytes to write.

ByteOffset: The absolute byte offset in the stream where data should be written. **ByteOffset** could be negative, which means the write should occur at the end of the stream.

ByteCount: The number of bytes in **InputBuffer** to write.

On completion, the object store MUST return:

Status: An NTSTATUS code that specifies the result.

BytesWritten: The number of bytes written.

Pseudocode for the operation is as follows:

If **Open.Mode.FILE_NO_INTERMEDIATE_BUFFERING** is TRUE and (**ByteOffset** >= 0), the operation MUST be failed with STATUS_INVALID_PARAMETER under any of the following conditions:

If (**ByteOffset** % **Open.File.Volume.LogicalBytesPerSector**) is not zero.

If (**ByteCount** % **Open.File.Volume.LogicalBytesPerSector**) is not zero.

If **ByteOffset** equals -2, then set ByteOffset to **Open.CurrentByteOffset**.

If **Open.File.Volume.IsReadOnly**, the operation MUST be failed with STATUS_MEDIA_WRITE_PROTECTED.

If ((**ByteOffset** + **ByteCount**) > MAXLONGLONG (0x7fffffffffffffff) and (**ByteOffset** >= 0), the operation MUST be failed with STATUS_INVALID_PARAMETER.

If **ByteCount** is zero, the object store MUST return:

BytesWritten set to 0.

Status set to STATUS_SUCCESS.

If ((**ByteOffset** < 0) and (**Open.Stream.Size + ByteCount))** > MAXLONGLONG (0x7fffffffffffffff), the operation MUST fail with STATUS_INVALID_PARAMETER.

If (**ByteOffset** < 0), set **ByteOffset** to **Open.Stream.Size**.

If (**ByteOffset** + **ByteCount**) > MAXFILESIZE (0xfffffff0000), the operation MUST be failed with STATUS_INVALID_PARAMETER.

Initialize *UsnReason* to zero.

If (**ByteOffset** + **ByteCount**) > **Open.Stream.Size**, set *UsnReason*.USN_REASON_DATA_EXTEND to TRUE.

If **ByteOffset** < **Open.Stream.Size**, set *UsnReason*.USN_REASON_DATA_OVERWRITE to TRUE.

If **Open.Stream.Oplock** is not empty, the object store MUST check for an oplock break according to the algorithm in section 3.1.4.12, with input values as follows:

Open equal to this operation's **Open**

Oplock equal to **Open.Stream.Oplock**

Operation equal to "WRITE"

OpParams empty

Determine if the write is in conflict with an existing byte range lock on **Open.Stream** using the algorithm described in section 3.1.4.10 (with **ByteOffset** set to **ByteOffset**, **Length** set to **ByteCount**, **IsExclusive** set to TRUE, **LockIntent** set to FALSE and **Open** set to **Open**). If the algorithm returns TRUE, the operation MUST be failed with STATUS_FILE_LOCK_CONFLICT.

The object store MUST post a USN change as per section 3.1.4.11 with **File** equal to **File**, **Reason** equal to *UsnReason*, and **FileName** equal to **Open.Link.Name**.

If ((**ByteOffset** + **ByteCount**) > **Open.Stream.ValidDataLength**), then set *DoingIoAtEof* to TRUE.

If ((**ByteOffset** + **ByteCount**) > **Open.Stream.AllocationSize**), the object store MUST increase **Open.Stream.AllocationSize** to ***BlockAlign***(**ByteOffset** + **ByteCount**, **Open.File.Volume.ClusterSize**). If there is not enough disk space, the operation MUST be failed with STATUS_DISK_FULL.

If **Open.Mode.FILE_NO_INTERMEDIATE_BUFFERING** is TRUE:

The object store MUST write any unwritten cached data for this range of the stream to disk.

The object store MUST remove from the cache any cached data for this range of the stream.

If *DoingIoAtEof* is TRUE, and (**Open.Stream.ValidDataLength** < **ByteOffset**) , write zeroes to the location on disk corresponding to the range between **Open.Stream.ValidDataLength** and **ByteOffset** in the stream, and then write the first **ByteCount** bytes of **InputBuffer** to the location on disk corresponding to the range starting at offset **ByteOffset** in the stream. If either write to the disk failed, the operation MUST be failed with the corresponding error status.

EndIf

If **Open.Mode.FILE_NO_INTERMEDIATE_BUFFERING** is FALSE, *DoingIoAtEof* is TRUE, and (**Open.Stream.ValidDataLength** < **ByteOffset**), zero out the range between **Open.Stream.ValidDataLength** and **ByteOffset** in the cache for this stream and then write the first **ByteCount** bytes of **InputBuffer** into the cache for this stream at offset **ByteOffset**. If there would not be enough disk space to flush the cache, the operation MUST be failed with STATUS_DISK_FULL. If **Open.Mode.FILE_WRITE_THROUGH** is TRUE, the cache write will also trigger a flush of the cache for that range to the disk.

If **Open.Mode.FILE_SYNCHRONOUS_IO_ALERT** is TRUE or **Open.Mode.FILE_SYNCHRONOUS_IO_NONALERT** is TRUE, the object store MUST set **Open.CurrentByteOffset** to (**ByteOffset** + **ByteCount**).

The object store MUST note that the file has been modified as specified in section 3.1.4.17 with **Open** equal to **Open**.

Upon successful completion of the operation, the object store MUST set:

Open.Stream.Size to the maximum of **Open.Stream.Size** or (**ByteOffset** + **ByteCount**).

Open.Stream.ValidDataLength to the maximum of **Open.Stream.ValidDataLength** or (**ByteOffset** + **ByteCount**).

BytesWritten to **ByteCount**.

Status to STATUS_SUCCESS.

3.1.5.4 Server Requests Closing an Open

Note: Some of the information in this section is subject to change because it applies to a preliminary implementation of the protocol or structure. For information about specific differences between versions, see the behavior notes that are provided in the Product Behavior appendix.

The server provides:

Open: The **Open** that the application is to close.

On completion, the object store MUST return:

Status: An NTSTATUS code that specifies the result.

This operation uses the following local variables:

Boolean values (initialized to FALSE): *LinkDeleted*, *StreamDeleted*, *FileDeleted*, *PostUsnClose*

The **Open** provided by the application MUST be removed from **Open.File.OpenList**.

Pseudocode for the operation is as follows:

```
Phase 1 - Delete on Close:
If Open.Mode.FILE_DELETE_ON_CLOSE is TRUE:
    If Open.Stream.StreamType is DirectoryStream or Open.Stream.Name is empty:
        Open.Link.IsDeleted MUST be set to TRUE.
    Else:
        Open.Stream.IsDeleted MUST be set to TRUE.
    EndIf
EndIf
Phase 2 -Stream Deletion:
If Open.Stream.IsDeleted is TRUE and Open.File.OpenList does not contain any Opens on
Open.Stream (this is a close of the last Open to a stream that has been marked deleted), then:
    Open.Stream MUST be removed from Open.File.StreamList.
    If Open.Stream.IsSparse is TRUE, and there does not exist an ExistingStream in
    Open.File.StreamList such that ExistingStream.IsSparse is TRUE:
        The object store MUST set Open.File.FileAttributes.FILE_ATTRIBUTE_SPARSE_FILE to
        FALSE, indicating that no streams of the file are sparse.
        The object store MUST post a USN change as per section 3.1.4.11 with File equal to File,
        Reason equal to USN_REASON_STREAM_CHANGE | USN_REASON_BASIC_INFO_CHANGE,
        and FileName equal to Open.Link.Name.
    Else:
        The object store MUST post a USN change as per section 3.1.4.11 with File equal to File,
        Reason equal to USN_REASON_STREAM_CHANGE, and FileName equal to
        Open.Link.Name.
    EndIf
    StreamDeleted MUST be set to TRUE.
```

PostUsnClose MUST be set to TRUE.

EndIf

Phase 3 - File Deletion:

If **Open.Link.IsDeleted** is TRUE and there does not exist an *ExistingOpen* in **Open.File.OpenList** that has *ExistingOpen***.Link** equal to **Open.Link:**

Remove **Open.Link** from **Open.File.LinkList**.

Remove **Open.Link** from **Open.Link.ParentFile.DirectoryList**.

Set *LinkDeleted* to TRUE.

If **Open.File.LinkList** is empty:

Set *FileDeleted* to TRUE.

EndIf

EndIf

Phase 4 - Truncate on Close:

Set *AllocationClusters* to ***ClustersFromBytes***(**Open**.**File**.**Volume**, **Open**.**Stream**.**AllocationSize**).

Set *FileClusters* to ***ClustersFromBytes***(**Open**.**File**.**Volume**, **Open**.**Stream**.**FileSize**).

If *AllocationClusters* > *FileClusters*:

This file has excess allocation. The object store SHOULD free (*AllocationClusters* - *FileClusters*) clusters of allocation from the end of the stream, and set **Open**.**Stream**.**AllocationSize** to *FileClusters* * **Open**.**File**.**Volume**.**ClusterSize**.

EndIf

Phase 5 -- Directory Change Notification:

When a directory **Open** with outstanding directory change notification requests is closed, these requests are completed using the algorithm below.

If **Open.Stream.StreamType** is DirectoryStream:

For each **ChangeNotifyEntry** in **Volume.ChangeNotifyList** where **ChangeNotifyEntry.OpenedDirectory** is equal to **Open** then the following actions MUST be taken:

Remove **ChangeNotifyEntry** from **Volume.ChangeNotifyList**.

Complete the **ChangeNotify** operation with status STATUS_NOTIFY_CLEANUP.

EndFor

EndIf

If **Open.Link** is deleted, a directory change notification on **Open.Link.ParentFile** MUST be issued. Pseudocode for these notifications is as follows:

If *LinkDeleted* is TRUE:

Set *Action* to FILE_ACTION_REMOVED.

If **Open.Stream.StreamType** is DirectoryStream:

Set *FilterMatch* to FILE_NOTIFY_CHANGE_DIR_NAME.

Else:

Set *FilterMatch* to FILE_NOTIFY_CHANGE_FILE_NAME.

EndIf

Send directory change notification as per section 3.1.4.1 with **Volume** equal to **Open.File.Volume**, **Action** equal to *Action*, **FilterMatch** equal to *FilterMatch*, and **FileName** equal to **Open.FileName**.

EndIf

If **Open.Stream** was deleted, then the stream deletion change notification MUST be issued. Pseudocode for this notification is as follows:

If *StreamDeleted* is TRUE:

Set *Action* to FILE_ACTION_REMOVED_STREAM.

Set *FilterMatch* to FILE_NOTIFY_CHANGE_STREAM_NAME.

Send directory change notification as per section 3.1.4.1 with **Volume** equal to **Open.File.Volume**, **Action** equal to *Action*, **FilterMatch** equal to *FilterMatch* and **FileName** equal to **Open.FileName**.

EndIf

If **Open.File** has had other changes that were not notified, a directory change notification reflecting those changes MUST be issued. Pseudocode for this notification is as follows:

Set *FilterMatch* to **Open.File.PendingNotifications**.

If *FilterMatch* is nonzero:

Set *Action* to FILE_ACTION_MODIFIED.

Send directory change notification as per section 3.1.4.1 with **Volume** equal to **Open.File.Volume**, **Action** equal to *Action*, **FilterMatch** equal to *FilterMatch* and **FileName** equal to **Open.FileName**.

Set **Open.File.PendingNotifications** to zero.

EndIf

If this is an **Open** to a named data **Stream** (**Open.Stream.StreamType** is DataStream and **Open.Stream.Name** is not empty) and there have been changes to it that weren't previously notified, a directory change notification reflecting those changes MUST be issued. Pseudocode for this notification is as follows:

Set *FilterMatch* to **Open.Stream.PendingNotifications**.

If *FilterMatch* is nonzero:

Set *Action* to FILE_ACTION_MODIFIED_STREAM.

Send directory change notification as per section 3.1.4.1 with **Volume** equal to **Open.File.Volume**, **Action** equal to *Action*, **FilterMatch** equal to *FilterMatch* and **FileName** equal to **Open.FileName**.

Set **Open.Stream.PendingNotifications** to zero.

EndIf

If *LinkDeleted* is TRUE:

If *FileDeleted* is FALSE:

Post a USN change as per section 3.1.4.11 with **File** equal to **File**, **Reason** equal to USN_REASON_HARD_LINK_CHANGE, and **FileName** equal to **Open.Link.Name**.

Set *PostUsnClose* to TRUE.

Else:

Post a USN change as per section 3.1.4.11 with **File** equal to **File**, **Reason** equal to USN_REASON_FILE_DELETE | USN_REASON_CLOSE, and **FileName** equal to **Open.Link.Name**.

EndIf

EndIf

Phase 6 -- USN Journal:

If *PostUsnClose* is TRUE:

Post a USN change as per section 3.1.4.11 with **File** equal to **File**, **Reason** equal to USN_REASON_CLOSE, and **FileName** equal to **Open.Link.Name**.

EndIf

Phase 7 -- Tunnel Cache:

If *LinkDeleted* is TRUE, then a new **TunnelCacheEntry** object *TunnelCacheEntry* MUST be constructed and added to the **Open.File.Volume.TunnelCacheList** as follows:

TunnelCacheEntry.**EntryTime** MUST be set to the current time.

TunnelCacheEntry.**ParentFile** MUST be set to **Open.Link.ParentFile**.

TunnelCacheEntry.**FileName** MUST be set to **Open.Link.Name**.

TunnelCacheEntry.**FileShortName** MUST be set to **Open.Link.ShortName**.

If **Open.FileName** matches **Open.Link.ShortName** then *TunnelCacheEntry*.**KeyByShortName** MUST be set to TRUE, else *TunnelCacheEntry*.**KeyByShortName** MUST be set to FALSE.

TunnelCacheEntry.**FileCreationTime** MUST be set to **Open.File.CreationTime**.

TunnelCacheEntry.**FileObjectId** MUST be set to **Open.File.ObjectId**.

EndIf

If **Open.File.FileType** is DirectoryFile and *LinkDeleted* is TRUE, then **Open.File** MUST have every *TunnelCacheEntry* associated with it invalidated:

For every *ExistingTunnelCacheEntry* in **Open.File.Volume.TunnelCacheList**:

If *ExistingTunnelCacheEntry*.**ParentFile** matches **Open.File**, then *ExistingTunnelCacheEntry* MUST be removed from **Open.File.Volume.TunnelCacheList**.

EndFor

EndIf

Phase 8 -- Oplock Cleanup:

If **Open.Stream.Oplock** is not empty, the object store MUST check for an oplock break according to the algorithm in section 3.1.4.12, with input values as follows:

Open equal to this operation's **Open**

Oplock equal to **Open.Stream.Oplock**

Operation equal to "CLOSE"

OpParams empty

If *LinkDeleted* is TRUE or *FileDeleted* is TRUE:

If the **Oplock** member of the **DirectoryStream** in **Open.Link.ParentFile.StreamList** (hereinafter referred to as *ParentOplock*) is not empty, the object store MUST check for an oplock break on the parent according to the algorithm in section 3.1.4.12, with input values as follows:

Open equal to this operation's **Open**

Oplock equal to *ParentOplock*

Operation equal to "CLOSE"

Flags equal to "PARENT_OBJECT"

EndIf

Phase 9 -- Byte Range Locks:

All elements from **Open.Stream.ByteRangeLockList** where **ByteRangeLock.OwnerOpen** == **Open** MUST be removed.

Phase 10 - Update Timestamps

If *LinkDeleted* is TRUE and *FileDeleted* is FALSE:

If **Open.UserSetChangeTime** is FALSE, update **Open.File.LastChangeTime** to the current time.

Set **Open.File.FileAttributes**.FILE_ATTRIBUTE_ARCHIVE to TRUE.

EndIf

If **Open**.**GrantedAccess**.FILE_EXECUTE is TRUE and **Open.UserSetAccessTime** is FALSE:

Update **Open.File.LastAccessTime** to the current time.

EndIf

Upon successful completion of this operation, the object store MUST return:

Status set to STATUS_SUCCESS.

3.1.5.5 Server Requests Querying a Directory

The server provides:

Open: An **Open** of a DirectoryStream.

FileInformationClass: The type of information being queried, as specified in [MS-FSCC] section 2.4.

OutputBufferSize: The maximum number of bytes to return in **OutputBuffer**.

RestartScan: A Boolean value which, if TRUE, indicates that enumeration should be restarted from the beginning of the directory. If FALSE, enumeration should continue from the last position.

ReturnSingleEntry: A Boolean value which, if TRUE, indicates that at most one entry MUST be returned. If FALSE, a variable count of entries could be returned, not to exceed **OutputBufferSize** bytes.

FileIndex: An index number from which to resume the enumeration if the object store supports it (optional).

FileNamePattern: A Unicode string containing the file name pattern to match. The object store MUST treat any asterisk ("*") and question mark ("?") characters in **FileNamePattern** as wildcards. **FileNamePattern** could be empty. The object store MUST treat an empty value as equivalent to the pattern "*".

On completion, the object store MUST return:

Status: An NTSTATUS code that specifies the result.

OutputBuffer: An array of bytes containing the query results. The structure of these bytes is dependent on the **FileInformationClass**, as noted in the relevant subsection.

ByteCount: The number of bytes stored in **OutputBuffer**.

3.1.5.5.1 FileObjectIdInformation

The following local variable is used:

Boolean value (initialized to FALSE): *EmptyPattern*

Support for this operation is optional. If the object store does not implement this functionality, the operation MUST be failed with STATUS_INVALID_DEVICE_REQUEST.

OutputBuffer is an array of one or more FILE_OBJECTID_INFORMATION structures as specified in [MS-FSCC] section 2.4.28.

This Information class can only be sent to a specific directory that maintains a list of all ObjectIDs on the volume. The name of this directory is: "\$Extend\$ObjId:$O:$INDEX_ALLOCATION". If it is sent to any other file or directory on the volume, the operation MUST be failed with STATUS_INVALID_INFO_CLASS.<38>

Pseudocode for the operation is as follows:

If **FileNamePattern** is not empty and **FileNamePattern.Length** (0 is a valid length) is not a multiple of 4, the operation MUST be failed with STATUS_INVALID_PARAMETER.

If **FileNamePattern** is empty, the object store MUST set *EmptyPattern* to TRUE; otherwise it MUST set *EmptyPattern* to FALSE.

If **FileNamePattern.Length** is less than the size of an ObjectId (16 bytes), **FileNamePattern.Buffer** will be zero filled up to the size of ObjectId.

The object store MUST search the volume for *Files* having *File*.**ObjectId** matching **FileNamePattern.** To determine if there is a match, **FileNamePattern.Buffer** is compared to **ObjectId** in chunks of ULONG (4 bytes). Any comparison where the **ObjectId** chunk is greater than or equal to the **FileNamePattern.Buffer** chunk is considered a match. If **FileNamePattern.Length** is longer than the size of **ObjectId** and the first 16 bytes (size of **ObjectId**) of **FileNamePattern.Buffer** is identical to *ObjectId*, **FileNamePatter.Buffer** is considered as greater than **ObjectId**.<39>

If **RestartScan** is FALSE and *EmptyPattern* is TRUE and there is no match, the operation MUST be failed with STATUS_NO_MORE_FILES.

The operation MUST fail with STATUS_NO_SUCH_FILE under any of the following conditions:

EmptyPattern is FALSE and there is no match.

EmptyPattern is TRUE and **RestartScan** is TRUE and there is no match.

The operation MUST fail with STATUS_BUFFER_OVERFLOW if **OutputBufferSize** < sizeof(FILE_OBJECTID_INFORMATION).

If there is at least one match, the operation is considered successful. The object store MUST return:

Status set to STATUS_SUCCESS.

OutputBuffer containing an array of as many FILE_OBJECTID_INFORMATION structures that match the query as will fit in **OutputBuffer** unless **ReturnSingleEntry** is TRUE, in which case only a single entry will be stored in **OutputBuffer**. To continue the query, **FileNamePattern** MUST be empty and RestartScan MUST be FALSE.

ByteCount set to the number of bytes filled in **OutputBuffer**.

3.1.5.5.2 FileReparsePointInformation

The following local variable is used:

Boolean value (initialized to FALSE): *EmptyPattern*

Support for this operation is optional. If the object store does not implement this functionality, the operation MUST be failed with STATUS_INVALID_DEVICE_REQUEST.

OutputBuffer is an array of one or more FILE_REPARSE_POINT_INFORMATION structures as specified in [MS-FSCC] section 2.4.35.

This Information class can only be sent to a specific directory that maintains a list of all Reparse Points on **Open.File.Volume**. The name of this directory is: "\$Extend\$Reparse:$R:$INDEX_ALLOCATION". If it is sent to any other file or directory on **Open.File.Volume**, the operation MUST be failed with STATUS_INVALID_INFO_CLASS.<40>

Pseudocode for the operation is as follows:

If **FileNamePattern** is not empty and **FileNamePattern.Length** (0 is a valid length) is not a multiple of 4, the operation MUST be failed with STATUS_INVALID_PARAMETER.

If **FileNamePattern** is empty, the object store MUST set *EmptyPattern* to TRUE; otherwise it MUST set *EmptyPattern* to FALSE.

If **FileNamePattern.Length** is less than the size of a **ReparseTag** (4 bytes), **FileNamePattern.Buffer** will be zero filled up to the size of ReparseTag.

If *EmptyPattern* is FALSE:

The object store MUST search **Open.File.Volume** for *Files* having *File* **ReparseTag** matching **FileNamePattern.**

Else

The object store MUST match all reparse tags on the volume.

EndIf

If **RestartScan** is FALSE and *EmptyPattern* is TRUE and there is no match, the operation MUST be failed with STATUS_NO_MORE_FILES.

The operation MUST fail with STATUS_NO_SUCH_FILE under any of the following conditions:

EmptyPattern is FALSE and there is no match.

EmptyPattern is TRUE and **RestartScan** is TRUE and there is no match.

The operation MUST fail with STATUS_BUFFER_OVERFLOW if **OutputBuffer** is not large enough to hold the first matching entry.

If there is at least one match, the operation is considered successful. The object store MUST return:

Status set to STATUS_SUCCESS.

OutputBuffer containing an array of as many FILE_REPARSE_POINT_INFORMATION structures that match the query as will fit in **OutputBuffer** unless **ReturnSingleEntry** is TRUE, in which case only a single entry will be stored in **OutputBuffer**. To continue the query, **Fi(_)-3(I)8(N)5(f)5(i)-50iPARS**

□

3.1.5.5.3 Directory Information Queries

This section describes how the object store processes directory queries for the following **FileInformationClass** values:

FileBothDirectoryInformation

FileDirectoryInformation

FileFullDirectoryInformation

FileIdBothDirectoryInformation

FileIdFullDirectoryInformation

FileNamesInformation

This algorithm uses the following local variables:

Boolean value (initialized to FALSE): *FirstQuery*

Link: *Link*

Stream: *DefaultStream*

32-bit Unsigned integers: *FileNameBytesToCopy*, *BaseLength*, *FoundNameLength*

Pointer to given **FileInformationClass** Structure: *Entry*, *LastEntry*

Status (initialized to STATUS_SUCCESS): *StatusToReturn*

Pseudocode for the algorithm is as follows:

If **OutputBufferSize** is less than the size needed to return a single entry, the operation MUST be failed with STATUS_INFO_LENGTH_MISMATCH. The below subsections describe the initial size checks for **OutputBufferSize** to determine whether any entries can be returned.

If **Open.File** is not a **DirectoryFile**, the operation MUST be failed with STATUS_INVALID_PARAMETER.

If **Open.QueryPattern** is empty:

If **FileNamePattern** is empty:

Set **FileNamePattern** to "*".

Else:

If **FileNamePattern** is not a valid filename component as described in [MS-FSCC] section 2.1.5, with the exceptions that wildcard characters described in section 3.1.4.3 are permitted and the strings "." and ".." are permitted, the operation MUST be failed with STATUS_OBJECT_NAME_INVALID.

EndIf

FirstQuery = TRUE

Set **Open.QueryPattern** to **FileNamePattern** for use in subsequent queries.

Else:

FirstQuery = FALSE

EndIf

If **RestartScan** is TRUE or **Open.QueryLastEntry** is empty:

Set **Open.QueryLastEntry** to the first *Link* in **Open.File.DirectoryList**, thus enumerating the directory from its beginning.

EndIf

Set *Entry* and *LastEntry* to point to the front of **OutputBuffer**.

Set **ByteCount** to zero.

Set *BaseLength* to ***FieldOffset(*FileInformationClass.FileName*)***. In other words save the size of the fixed length portion of the given Information Class.

For each *Link* in **Open.File.DirectoryList** starting at **Open.QueryLastEntry**:

If **ReturnSingleEntry** is TRUE and *Entry* != **OutputBuffer**, then break.

If *FirstQuery* is TRUE, the object store MUST set the "." and ".." file names as the first two records returned unless one of the following is TRUE:

Open.File == **File.Volume.RootDirectory**

FileNamePattern == "."

FileNamePattern contains wildcard characters as described in section 3.1.4.3 and the Unicode string "." matches **FileNamePattern** according to the algorithm in section 3.1.4.4.

EndIf

If *Link*.**Name** or *Link*.**ShortName** matches **FileNamePattern** as described in section 3.1.4.4 using the following parameters: **FileName** set to *Link*.**Name** then *Link*.**ShortName** if not empty, **Expression** set to **FileNamePattern** and **Ignorecase** set to **Open.IsCaseInsensitive**, then:

Set *FoundNameLength* to the length, in bytes, of *Link*.Name.

If *Entry* != ***OutputBuffer(***one or more structures have already been copied into **OutputBuffer*)*** and (**ByteCount** + *BaseLength* + *FoundNameLength*) > **OutputBufferSize** then break.

Set *DefaultStream* to the entry in *Link*.**File.StreamList** where *DefaultStream*.**Name** is empty (locate the default stream for the given file or directory).

The object store MUST copy the fixed portion of the given **FileInformationClass** structure to *Entry* as described in the subsections below. This does not include copying the **FileName** field.

If (**ByteCount** + *BaseLength* + *FoundNameLength*) > **OutputBufferSize** then:

Set *FileNameBytesToCopy* to **OutputBufferSize** - **ByteCount** - *BaseLength*.

Set *StatusToReturn* to STATUS_BUFFER_OVERFLOW.

The scenario where a partial filename is returned only occurs on the first record being returned. The earlier checks guarantee that there will be room for the fixed portion of the given **FileInformationClass** structure.

EndIf

Copy *FileNameBytesToCopy* bytes from *Link***.Name** into **FileInformationClass.Filename** field.

Set *LastEntry***.NextEntryOffset** to *Entry* - **OutputBuffer**.

Set **ByteCount** to **BlockAlign(ByteCount**, 8**)** + *BaseLength* + *FileNameBytesToCopy*.

If *StatusToReturn* != STATUS_SUCCESS, then break.

Set *LastEntry* to *Entry*.

Set *Entry* to **OutputBuffer** + **ByteCount**, which points to the beginning of the next record to be returned (if any).

EndIfSet **Open.QueryLastEntry** to *Link*.

EndFor

If no records are being returned:

If *FirstQuery* is TRUE:

Set *StatusToReturn* to STATUS_NO_SUCH_FILE, which means no files were found in this directory that match the given wildcard pattern.

Else:

Set *StatusToReturn* to STATUS_NO_MORE_FILES, which means no more files were found in this directory that match the given wildcard pattern.

EndIf

If **Open.File.UserSetAccessTime** is FALSE, the object store MUST update **Open.File.LastAccessTime** to the current system time.

The object store MUST return:

Status set to *StatusToReturn*.

OutputBuffer containing an array of as many entries that match the query as will fit in **OutputBufferSize**.

BytesReturned containing the number of bytes filled in **OutputBuffer**.

3.1.5.5.3.1 FileBothDirectoryInformation

OutputBuffer is an array of one or more FILE_BOTH_DIR_INFORMATION structures as described in [MS-FSCC] section 2.4.8. *Entry* is a parameter to this routine that points to the current FILE_BOTH_DIR_INFORMATION structure to fill out. Note that the FileName field is not set in this section.

Pseudocode for the operation is as follows:

If **OutputBufferSize** is smaller than ***FieldOffset(***FILE_BOTH_DIR_INFORMATION.FileName***)***, the operation MUST be failed with STATUS_INFO_LENGTH_MISMATCH.

The object store MUST process this query using the algorithm described in section 3.1.5.5.3.

Entry MUST be filled out as follows:

Entry.**NextEntryOffset** set to zero

Entry.**FileIndex** set to zero

Entry.**CreationTime** set to *Link*.**File.CreationTime**

Entry.**LastAccessTime** set to *Link*.**File.LastAccessTime**

Entry.**LastWriteTime** set to *Link*.**File.LastModificationTime**

Entry.**ChangeTime** set to *Link*.**File.LastChangeTime**

Entry.**EndOfFile** set to *DefaultStream*.**Size**

Entry.**AllocationSize** set to *DefaultStream*.**AllocationSize**

Entry.**FileAttributes** set to *Link*.**File.FileAttributes**

If *Link*.**File.FileType** is DirectoryFile:

Entry.**FileAttributes.FILE_ATTRIBUTE_DIRECTORY** is set

EndIf

If *Entry*.**FileAttributes** has no attributes set:

Entry.**FileAttributes.FILE_ATTRIBUTE_NORMAL** is set

EndIf

If *Link*.**File.FileAttributes.FILE_ATTRIBUTE_REPARSE_POINT** is set:

Entry.**EaSize** set to *Link*.**File.ReparseTag**

Else:

Entry.**EaSize** set to *Link*.**File.ExtendedAttributesLength**<41>

EndIf

If *Link*.**ShortName** is not empty:

Entry.**ShortNameLength** set to the length, in bytes, of *Link*.**ShortName**

Entry.**ShortName** set to *Link*.**ShortName** padding with zeroes as necessary

Else:

Entry.**ShortNameLength** set to zero

Entry.**ShortName** is filled with zeroes

EndIf

Entry.**FileNameLength** set to the length ,in bytes, of *Link*.**Name**

3.1.5.5.3.2 FileDirectoryInformation

OutputBuffer is an array of one or more FILE_DIRECTORY_INFORMATION structures as described in [MS-FSCC] section 2.4.10. *Entry* is a parameter to this routine that points to the current FILE_DIRECTORY_INFORMATION structure to fill out. Note that the FileName field is not set in this section.

Pseudocode for the operation is as follows:

If **OutputBufferSize** is smaller than ***FieldOffset(***FILE_DIRECTORY_INFORMATION.FileName***)***, the operation MUST be failed with STATUS_INFO_LENGTH_MISMATCH.

The object store MUST process this query using the algorithm described in section 3.1.5.5.3.

Entry MUST be filled out as follows:

Entry.**NextEntryOffset** set to zero

Entry.**FileIndex** set to zero

Entry.**CreationTime** set to *Link*.**File.CreationTime**

Entry.**LastAccessTime** set to *Link*.**File.LastAccessTime**

Entry.**LastWriteTime** set to *Link*.**File.LastModificationTime**

Entry.**ChangeTime** set to *Link*.**File.LastChangeTime**

Entry.**EndOfFile** set to *DefaultStream*.**Size**

Entry.**AllocationSize** set to *DefaultStream*.**AllocationSize**

Entry.**FileAttributes** set to *Link*.**File.FileAttributes**

If *Link*.**File.FileType** is DirectoryFile:

Entry.**FileAttributes.FILE_ATTRIBUTE_DIRECTORY** is set

EndIf

If *Entry*.**FileAttributes** has no attributes set:

Entry.**FileAttributes.FILE_ATTRIBUTE_NORMAL** is set

EndIf

Entry.**FileNameLength** set to the length ,in bytes, of *Link*.**Name**

3.1.5.5.3.3 FileFullDirectoryInformation

OutputBuffer is an array of one or more FILE_FULL_DIR_INFORMATION structures as described in [MS-FSCC] section 2.4.14. *Entry* is a parameter to this routine that points to the current FILE_FULL_DIR_INFORMATION structure to fill out. Note that the FileName field is not set in this section.

Pseudocode for the operation is as follows:

If **OutputBufferSize** is smaller than ***FieldOffset(***FILE_FULL_DIR_INFORMATION.FileName***)***, the operation MUST be failed with STATUS_INFO_LENGTH_MISMATCH.

The object store MUST process this query using the algorithm described in section 3.1.5.5.3.

Entry MUST be filled out as follows:

Entry.**NextEntryOffset** set to zero

Entry.**FileIndex** set to zero

Entry.**CreationTime** set to *Link*.**File.CreationTime**

Entry.**LastAccessTime** set to *Link*.**File.LastAccessTime**

Entry.**LastWriteTime** set to *Link*.**File.LastModificationTime**

Entry.**ChangeTime** set to *Link*.**File.LastChangeTime**

Entry.**EndOfFile** set to *DefaultStream*.**Size**

Entry.**AllocationSize** set to *DefaultStream*.**AllocationSize**

Entry.**FileAttributes** set to *Link*.**File.FileAttributes**

If *Link*.**File.FileType** is DirectoryFile:

Entry.**FileAttributes.FILE_ATTRIBUTE_DIRECTORY** is set

EndIf

If *Entry*.**FileAttributes** has no attributes set:

Entry.**FileAttributes.FILE_ATTRIBUTE_NORMAL** is set

EndIf

If *Link*.**File.FileAttributes.FILE_ATTRIBUTE_REPARSE_POINT** is SET:

Entry.**EaSize** set to *Link*.**File.ReparseTag**

Else:

Entry.**EaSize** set to *Link*.**File.ExtendedAttributesLength**<42>

EndIf

Entry.**FileNameLength** set to the length, in bytes, of *Link*.**Name**

3.1.5.5.3.4 FileIdBothDirectoryInformation

OutputBuffer is an array of one or more FILE_ID_BOTH_DIR_INFORMATION structures as described in [MS-FSCC] section 2.4.17. *Entry* is a parameter to this routine that points to the current FILE_ID_BOTH_DIR_INFORMATION structure to fill out. Note that the FileName field is not set in this section.

Pseudocode for the operation is as follows:

If **OutputBufferSize** is smaller than ***FieldOffset(***FILE_ID_BOTH_DIR_INFORMATION.FileName***)***, the operation MUST be failed with STATUS_INFO_LENGTH_MISMATCH.

The object store MUST process this query using the algorithm described in section 3.1.5.5.3.

Entry MUST be filled out as follows:

Entry.**NextEntryOffset** set to zero

Entry.**FileIndex** set to zero

Entry.**CreationTime** set to *Link*.**File.CreationTime**

Entry.**LastAccessTime** set to *Link*.**File.LastAccessTime**

Entry.**LastWriteTime** set to *Link*.**File.LastModificationTime**

Entry.**ChangeTime** set to *Link*.**File.LastChangeTime**

Entry.**EndOfFile** set to *DefaultStream*.**Size**

Entry.**AllocationSize** set to *DefaultStream*.**AllocationSize**

Entry.**FileAttributes** set to *Link*.**File.FileAttributes**

If *Link*.**File.FileType** is DirectoryFile:

Entry.**FileAttributes.FILE_ATTRIBUTE_DIRECTORY** is set

EndIf

If *Entry*.**FileAttributes** has no attributes set:

Entry.**FileAttributes.FILE_ATTRIBUTE_NORMAL** is set

EndIf

If *Link*.**File.FileAttributes.FILE_ATTRIBUTE_REPARSE_POINT** is SET:

Entry.**EaSize** set to *Link*.**File.ReparseTag**

Else:

Entry.**EaSize** set to *Link*.**File.ExtendedAttributesLength**<43>

EndIf

If *Link*.**ShortName** is not empty:

Entry.**ShortNameLength** set to the length, in bytes, of *Link*.**ShortName**

Entry.**ShortName** set to *Link*.**ShortName** padding with zeroes as necessary

Else:

Entry.**ShortNameLength** set to zero

Entry.**ShortName** filled with zeroes

EndIf

Entry.**FileID** set to *Link*.**File.FileID**

Entry.**FileNameLength** set to the length, in bytes, of *Link*.**Name**

3.1.5.5.3.5 FileIdFullDirectoryInformation

OutputBuffer is an array of one or more FILE_ID_FULL_DIR_INFORMATION structures as described in [MS-FSCC] section 2.4.18. *Entry* is a parameter to this routine that points to the current FILE_ID_FULL_DIR_INFORMATION structure to fill out. Note that the FileName field is not set in this section.

Pseudocode for the operation is as follows:

If **OutputBufferSize** is smaller than ***FieldOffset(***FILE_ID_FULL_DIR_INFORMATION.FileName***)***, the operation MUST be failed with STATUS_INFO_LENGTH_MISMATCH.

The object store MUST process this query using the algorithm described in section 3.1.5.5.3.

Entry MUST be filled out as follows:

Entry.**NextEntryOffset** set to zero

Entry.**FileIndex** set to zero

Entry.**CreationTime** set to *Link*.**File.CreationTime**

Entry.**LastAccessTime** set to *Link*.**File.LastAccessTime**

Entry.**LastWriteTime** set to *Link*.**File.LastModificationTime**

Entry.**ChangeTime** set to *Link*.**File.LastChangeTime**

Entry.**EndOfFile** set to *DefaultStream*.**Size**

Entry.**AllocationSize** set to *DefaultStream*.**AllocationSize**

Entry.**FileAttributes** set to *Link*.**File.FileAttributes**

If *Link*.**File.FileType** is DirectoryFile:

Entry.**FileAttributes.FILE_ATTRIBUTE_DIRECTORY** is set

EndIf

If *Entry*.**FileAttributes** has no attributes set:

Entry.**FileAttributes.FILE_ATTRIBUTE_NORMAL** is set

EndIf

If *Link*.**File.FileAttributes.FILE_ATTRIBUTE_REPARSE_POINT** is SET:

Entry.**EaSize** set to *Link*.**File.ReparseTag**

Else:

Entry.**EaSize** set to *Link*.**File.ExtendedAttributesLength**<44>

EndIf

Entry.**FileID** set to *Link*.**File.FileID**

Entry.**FileNameLength** set to the length, in bytes, of *Link*.**Name**

3.1.5.5.3.6 FileNamesInformation

OutputBuffer is an array of one or more FILE_NAMES_INFORMATION structures as described in [MS-FSCC] section 2.4.26. *Entry* is a parameter to this routine that points to the current FILE_NAMES_INFORMATION structure to fill out. Note that the FileName field is not set in this section.

Pseudocode for the operation is as follows:

If **OutputBufferSize** is smaller than ***FieldOffset(***FILE_NAMES_INFORMATION.FileName***)***, the operation MUST be failed with STATUS_INFO_LENGTH_MISMATCH.

The object store MUST process this query using the algorithm described in section 3.1.5.5.3.

Entry MUST be filled out as follows:

Entry.**NextEntryOffset** set to zero

Entry.**FileIndex** set to zero

Entry.**FileNameLength** set to the length, in bytes, of *Link*.**Name**

3.1.5.6 Server Requests Flushing Cached Data

The server provides:

Open: An **Open** of a DataFile or DirectoryFile for which it is to flush cached data.

On completion, the object store MUST return:

Status: An NTSTATUS code that specifies the result.

The object store MUST flush all persistent attributes for **Open.File** to stable storage. In addition:

If **Open.File.Volume.IsReadOnly** is TRUE, the operation MUST be failed with STATUS_MEDIA_WRITE_PROTECTED.

The operation MUST be failed with the status code returned from the underlying physical storage. The operation flushes all eligible objects; however, only the first failure encountered is returned.

The operation ensures that the directory structure is persisted to stable storage.<45>

Pseudocode for the operation is as follows:

If **Open.FileType** is DirectoryFile:

CurrentDirectory = **Open.DirectoryFile**

Flush *CurrentDirectory*

While *CurrentDirectory* != *CurrentDirectory*.**Volume.RootDirectory**:

Set *CurrentLink* to the head of *CurrentDirectory*.**LinkList**, which should be the only link because directories cannot have hard links.

CurrentDirectory = *CurrentLink*.**ParentFile**

Flush *CurrentDirectory*

EndWhile

EndIf

Flush all open objects on the volume.

If **Open.File** is equal to **Open.File.Volume.RootDirectory**:

For each *OpenFile* in **Open.File.Volume.OpenFileList**:

Flush *OpenFile*

EndFor

EndIf

3.1.5.7 Server Requests a Byte-Range Lock

The server provides:

Open: An **Open** of a DataStream.

FileOffset: A 64-bit unsigned integer containing the starting offset, in bytes.

Length: A 64-bit unsigned integer containing the length, in bytes. This value MAY be zero.

ExclusiveLock: A Boolean indicating whether the range is to be locked exclusively (TRUE) or shared (FALSE).

FailImmediately: A Boolean indicating whether the lock request is to fail (TRUE) if the range is locked by another open or if it is to wait until the lock can be acquired (FALSE).

On completion, the object store MUST return:

Status: An NTSTATUS code that specifies the result

Pseudocode for the operation is as follows:

[Validation]

If **Open.Stream.StreamType** is DirectoryStream, return STATUS_INVALID_PARAMETER, as byte range locks are not permitted on directories.

If (((**FileOffset** + **Length** - 1) < **FileOffset**) && **Length** != 0)

This means that the requested range contains one or more bytes with offsets beyond the maximum 64-bit unsigned integer. The operation MUST be failed with STATUS_INVALID_LOCK_RANGE.

EndIf

[Processing]

The object store MUST check for byte range lock conflicts by using the algorithm described in section 3.1.4.10, with **ByteOffset** set to **FileOffset**, **Length** set to **Length**, **IsExclusive** set to **ExclusiveLock**, **LockIntent** set to TRUE, and **Open** set to **Open**. If a conflict is detected, then:

If **FailImmediately** is TRUE, the operation MUST be failed with STATUS_LOCK_NOT_GRANTED.

Else

Insert operation into **CancelableOperations.CancelableOperationList**.

Wait until there are no overlapping **ByteRangeLocks** or until the operation is canceled per section 3.1.5.19. Overlapping **ByteRangeLocks** can be removed from **ByteRangeLockList** in different ways:

The **ByteRangeLock** can be explicitly unlocked as described in section 3.1.5.8.

The **ByteRangeLock.OwnerOpen** can be closed as described in section 3.1.5.4.

EndIf

EndIf

Initialize a new *ByteRangeLock*:

ByteRangeLock.**LockOffset** MUST be initialized to **FileOffset**.

ByteRangeLock.**LockLength** MUST be initialized to **Length**.

ByteRangeLock.**IsExclusive** MUST be initialized to **ExclusiveLock**.

ByteRangeLock.**OwnerOpen** MUST be initialized to **Open**.

Insert *ByteRangeLock* into **Open.Stream.ByteRangeLockList**.

Complete this operation with STATUS_SUCCESS.

3.1.5.8 Server Requests an Unlock of a Byte-Range

The server provides:

Open: An **Open** of a DataStream.

FileOffset: A 64-bit unsigned integer containing the starting offset, in bytes.

Length: A 64-bit unsigned integer containing the length, in bytes.

On completion, the object store MUST return:

Status: An NTSTATUS code that specifies the result.

Pseudocode for the operation is as follows:

[Validation]

If **Open.Stream.StreamType** is DirectoryStream, return STATUS_INVALID_PARAMETER, as byte range locks are not permitted on directories.

If (((**FileOffset** + **Length** - 1) < **FileOffset**) && **Length** != 0)

This means that the requested range contains one or more bytes with offsets beyond the maximum 64-bit unsigned integer. The operation MUST be failed with STATUS_INVALID_LOCK_RANGE.

EndIf

[Processing]

Initialize *LockToRemove* to NULL.

For each *ByteRangeLock* in **Open.Stream.ByteRangeLockList**:

If ((*ByteRangeLock*.**LockOffset** == **FileOffset**) and (*ByteRangeLock*.**LockLength** == **Length**) and (*ByteRangeLock*.**OwnerOpen** == **Open**)) then:

Set *LockToRemove* to *ByteRangeLock*.

If (*LockToRemove*.**ExclusiveLock** == TRUE) then break.

EndIf

EndFor

If *LockToRemove* is not NULL:

Remove *LockToRemove* from **Open.Stream.ByteRangeLockList**.

Complete this operation with STATUS_SUCCESS.

Else:

Complete this operation with STATUS_RANGE_NOT_LOCKED.

EndIf

3.1.5.9 Server Requests an FsControl Request

The following section describes various File System Control (FSCTLs) operations that are implemented by the Object Store. Not all of these operations are implemented by all file systems.

3.1.5.9.1 FSCTL_CREATE_OR_GET_OBJECT_ID

The server provides:

Open: An **Open** of a DataFile or DirectoryFile.

OutputBufferSize: The maximum number of bytes to return in **OutputBuffer**.

On completion, the object store MUST return:

Status: An NTSTATUS code that specifies the result.

OutputBuffer: An array of bytes that will return a FILE_OBJECTID_BUFFER structure as specified in [MS-FSCC] section 2.1.3.

BytesReturned: The number of bytes returned in **OutputBuffer**.

Support for this operation is optional. If the object store does not implement this functionality, the operation MUST be failed with STATUS_INVALID_DEVICE_REQUEST.<46>

Pseudocode for the operation is as follows:

If **Open.File.Volume.IsObjectIDsSupported** is FALSE, the operation MUST be failed with STATUS_VOLUME_NOT_UPGRADED.

If **OutputBufferSize** is less than ***sizeof(***FILE_OBJECTID_BUFFER***)***, the operation MUST be failed with STATUS_INVALID_PARAMETER.

If **Open.File.ObjectId** is empty:

If **Open.File.Volume.IsReadOnly**, the operation MUST be failed with STATUS_MEDIA_WRITE_PROTECTED.

The object store MUST set **Open.File.ObjectId** to a newly generated ObjectId GUID that is unique on **Open.File.Volume**.<47>

EndIf

If a new **Open.File.ObjectId** was generated above or if **Open.File.BirthVolumeId** and **Open.File.BirthObjectId** are both empty:

If **Open.File.Volume.IsReadOnly**, the operation MUST be failed with STATUS_MEDIA_WRITE_PROTECTED.

If **Open.File.BirthVolumeId** is empty, the object store MUST set **Open.File.BirthVolumeId** to **Open.File.Volume.VolumeId**.

If **Open.File.BirthObjectId** is empty, the object store MUST set **Open.File.BirthObjectId** to **Open.File.ObjectId**.

The object store MUST post a USN change as per section 3.1.4.11 with **File** equal to **File**, **Reason** equal to USN_REASON_OBJECT_ID_CHANGE, and **FileName** equal to **Open.Link.Name**.

EndIf

If a new **Open.File.ObjectId** was generated above, the object store MUST update **Open.File.LastChangeTime**.<48>

The object store MUST populate the fields of **OutputBuffer** as follows:

OutputBuffer.ObjectId set to **Open.File.ObjectId**.

OutputBuffer.BirthVolumeId set to **Open.File.BirthVolumeId**.

OutputBuffer.BirthObjectId set to **Open.File.BirthObjectId**.

OutputBuffer.DomainId set to empty.

Upon successful completion of the operation, the object store MUST return:

BytesReturned set to ***sizeof(***FILE_OBJECTID_BUFFER***)***.

Status set to STATUS_SUCCESS.

3.1.5.9.2 FSCTL_DELETE_OBJECT_ID

The server provides:

Open: An **Open** of a DataFile or DirectoryFile.

On completion, the object store MUST return:

Status: An NTSTATUS code that specifies the result.

Support for this operation is optional. If the object store does not implement this functionality, the operation MUST be failed with STATUS_INVALID_DEVICE_REQUEST.<49>

Pseudocode for the operation is as follows:

If **Open.File.Volume.IsObjectIDsSupported** is FALSE, the operation MUST be failed with STATUS_VOLUME_NOT_UPGRADED.

If **Volume.IsReadOnly** is TRUE, the operation MUST be failed with STATUS_MEDIA_WRITE_PROTECTED.

If **Open.File.ObjectId** is empty, the operation MUST be completed with STATUS_SUCCESS.

Update **Open.File.LastChangeTime** to the current time.<50>

Post a USN change as per section 3.1.4.11 with **File** equal to **File**, **Reason** equal to USN_REASON_OBJECT_ID_CHANGE, and **FileName** equal to **Open.Link.Name**.

Set **Open.File.ObjectId** to empty.

Upon successful completion of the operation, the object store MUST return:

Status set to STATUS_SUCCESS.

3.1.5.9.3 FSCTL_DELETE_REPARSE_POINT

The server provides:

Open: An **Open** of a DataFile or DirectoryFile.

ReparseTag: An identifier indicating the type of the reparse point to delete, as defined in [MS-FSCC] section 2.1.2.1.

ReparseGUID: A GUID indicating the type of the reparse point to delete.

On completion, the object store MUST return:

Status: An NTSTATUS code that specifies the result.

Support for this operation is optional. If the object store does not implement this functionality, the operation MUST be failed with STATUS_INVALID_DEVICE_REQUEST.<51>

Pseudocode for the operation is as follows:

Phase 1 -- Verify the parameters.

If (**Open.GrantedAccess** & (FILE_WRITE_DATA | FILE_WRITE_ATTRIBUTES)) == 0, the operation MUST be failed with STATUS_ACCESS_DENIED.

If **Open.File.Volume.IsReadOnly** is TRUE, the operation MUST be failed with STATUS_MEDIA_WRITE_PROTECTED.

If **Open.File.Volume.IsReparsePointsSupported** is FALSE, the operation MUST be failed with STATUS_VOLUME_NOT_UPGRADED.

If the **ReparseTag** is either IO_REPARSE_TAG_RESERVED_ZERO or IO_REPARSE_TAG_RESERVED_ONE, the operation MUST be failed with STATUS_IO_REPARSE_TAG_INVALID. The reserved reparse tags are defined in [MS-FSCC] section 2.1.2.1.

If **ReparseTag** is a non-Microsoft Reparse Tag, then the **ReparseGUID** MUST be a valid GUID; otherwise the operation MUST be failed with STATUS_IO_REPARSE_DATA_INVALID.

Phase 2 -- Validate that the requested tag deletion type matches with the stored tag type.

If (**ReparseTag** != **Open.File.ReparseTag**), the operation MUST be failed with STATUS_IO_REPARSE_TAG_MISMATCH.

If (**ReparseTag** is a non-Microsoft Reparse Tag && **Open.File.ReparseGUID** != **ReparseGUID**), the operation MUST be failed with STATUS_REPARSE_ATTRIBUTE_CONFLICT.

Phase 3 -- Remove the reparse point from the File.

Set **Open.File.ReparseData**, **Open.File.ReparseGUID**, and **Open.File.ReparseTag** to empty.

Update **Open.File.LastChangeTime** to the current system time.<52>

If **Open.File.FileType** == DataFile, set **Open.File.FileAttributes**.FILE_ATTRIBUTE_ARCHIVE to TRUE.

Set **Open.File.PendingNotifications**.FILE_NOTIFY_CHANGE_LAST_ACCESS to TRUE.

Upon successful completion of the operation, the object store MUST return:

Status set to STATUS_SUCCESS.

3.1.5.9.4 FSCTL_FILE_LEVEL_TRIM

The server provides:

Open: An **Open** of a DataFile.

InputBuffer: An array of bytes containing a single **FILE_LEVEL_TRIM** structure, followed by zero or more **FILE_LEVEL_TRIM_RANGE** structures, as specified in [MS-FSCC] section 2.3.69.1.

InputBufferSize: The number of bytes in **InputBuffer**.

OutputBufferSize: The number of bytes in **OutputBuffer**.

On completion, the object store MUST return:

Status: An NTSTATUS code that specifies the result.

OutputBuffer: An array of bytes that contains a single **FILE_LEVEL_TRIM_OUTPUT** structure, as specified in ([MS-FSCC] section 2.3.70).

BytesReturned: The number of bytes written to **OutputBuffer**.

This operation also uses the following local variables:

64-bit unsigned integers (initialized to zero): *AlignmentAdjust*, *TempOffLen*, *TrimRange*, *TrimOffset*.

An NTSTATUS code: *TrimStatus*.

Support for this operation is optional. If the object store does not implement this functionality, the operation MUST be failed with STATUS_INVALID_DEVICE_REQUEST.

Pseudocode for the operation is as follows:

If **Open.Stream.IsEncrypted** is TRUE OR **Open. Stream.IsCompressed** is TRUE, the operation MUST be failed with STATUS_INVALID_PARAMETER.

If **InputBuffer.Size** is < ***sizeof(***FILE_LEVEL_TRIM***)***, the operation MUST be failed with STATUS_INVALID_PARAMETER.

If **InputBuffer.NumRanges** is <= 0, the operation MUST be failed with STATUS_INVALID_PARAMETER.

If **InputBuffer.NumRanges** is – 1) * ***sizeof(***FILE_LEVEL_TRIM_RANGE***)*** overflows 32-bits, the operation MUST be failed with STATUS_INVALID_PARAMETER.

If **InputBuffer.NumRanges** – 1) * ***sizeof(***FILE_LEVEL_TRIM_RANGE***)***+ ***sizeof(***FILE_LEVEL_TRIM***)*** overflows 32-bits, the operation MUST be failed with STATUS_INVALID_PARAMETER.

If **OutputBufferSize** != 0 AND **OutputBufferSize** is < ***sizeof(***FILE_LEVEL_TRIM_OUTPUT***)***, the operation MUST be failed with STATUS_INVALID_PARAMETER.

If **Open.Volume.IsUsnJournalActive** is TRUE, the object store MUST post a USN change as per section 3.1.4.11 with File equal to **Open.File**, Reason equal to USN_REASON_DATA_OVERWRITE, and **FileName** equal to **Open.File.Name**.

Set **OutputBuffer.NumRangesProcessed** = 0.

For each *TrimRange* in **InputBuffer.Ranges:**

Set *TrimOffset* = **TrimRange.Offset**

Set *TrimLength* = **TrimRange.Length**

If ((*TrimOffset* % **Open.Volume.SystemPageSize**) != 0):

AlignmentAdjust = *TrimOffset* % **Open.Volume.SystemPageSize**

If (*TrimOffset* + **Open.Volume.SystemPageSize** –*AlignmentAdjust*) overflows 64-bits, the operation must be failed with STATUS_INTEGER_OVERFLOW.

If (*TrimLength* >= (**Open.Volume.SystemPageSize** –*AlignmentAdjust*):

Decrement *TrimLength* by (**Open.Volume.SystemPageSize** –*AlignmentAdjust*)

Else:

Set *TrimLength* to 0

EndIf

If (*TrimOffset* < **Open.File.EndOfFile**):

Set *TempOffLen* to *TrimOffset* + *TrimLength*

If **TempOffLen** overflows 64-bits, the operation MUST be failed with STATUS_INTEGER_OVERFLOW.

If *TempOffLen* > **Open.File.EndOfFile**:

TrimLength = **Open.File.EndOfFile** –*TrimOffset*

EndIf

EndIf

Decrement *TrimLength* by (*TrimLength* % **Open.Volume.SystemPageSize**)

If *TrimLength* == 0, skip further processing on this range and continue to the next range.

Construct a list of the LBAs that the object store denotes as the range of the file specified with *TrimOffset* and *TrimLength*. Send a TRIM command to the underlying storage device with the constructed list of LBAs. For ATA devices, this command is the T13 defined "TRIM". For SCSI/SAS devices, this command is the T10 defined "UNMAP". Store the status from the operation in *TrimStatus*.

If the command was successful:

Increment **OutputBuffer.NumRanges** by 1

Else,

The operation MUST return immediately with status set to *TrimStatus*.

EndIf

EndFor

Upon successful completion of the operation, the object store MUST return:

BytesReturned set to 0 If OutputBufferSize == 0, ***sizeof(***FILE_LEVEL_TRIM_OUTPUT***)*** otherwise

Status set to STATUS_SUCCESS.

3.1.5.9.5 FSCTL_FILESYSTEM_GET_STATISTICS

The server provides:

Open: An Open of a DataFile or DirectoryFile.

OutputBufferSize: The maximum number of bytes to return in **OutputBuffer**.

On completion, the object store MUST return:

Status: An NTSTATUS code that specifies the result.

OutputBuffer: An array of bytes that will return an array of statistical data, one entry per host processor.

BytesReturned: The number of bytes returned in **OutputBuffer**.

This operation also uses the following local variables:

An array of bytes (initially empty): *FileSystemStatistics*.

Support for this operation is optional. If the object store does not implement this functionality, the operation MUST be failed with STATUS_INVALID_DEVICE_REQUEST.<53>

Pseudocode for the operation is as follows:

If **OutputBufferSize** is less than sizeof(FILESYSTEM_STATISTICS), the operation is failed with STATUS_BUFFER_TOO_SMALL.

If **OutputBufferSize** is less than the total size of statistics information, then only **OutputBufferSize** bytes will be returned, and the operation MUST succeed but return with STATUS_BUFFER_OVERFLOW.

For each host processor, add one entry to *FileSystemStatistics* as follows:

FILESYSTEM_STATISTICS structure as specified in [MS-FSCC] section 2.3.8.1.

An optional file system-specific structure as specified in [MS-FSCC] section 2.3.8.2.<54>

Padding bytes of zeros to bring total size of each entry to be a multiple of 64 bytes.

EndFor

If **OutputBufferSize** is less than the total size of *FileSystemStatistics*, the object store MUST:

Copy **OutputBufferSize** bytes from *FileSystemStatistics* to **OutputBuffer**.

Set **BytesReturned** to the number of bytes copied to **OutputBuffer**.

Return **Status** set to STATUS_BUFFER_OVERFLOW.

EndIf

Upon successful completion of the operation, the object store MUST return:

Copy *FileSystemStatistics* to **OutputBuffer**.

Set **BytesReturned** to the number of bytes copied to **OutputBuffer**.

Return **Status** set to STATUS_SUCCESS.

3.1.5.9.6 FSCTL_FIND_FILES_BY_SID

The server provides:

Open: An **Open** of a DirectoryStream.

FindBySidData: An array of bytes containing a FIND_BY_SID_DATA structure as described in [MS-FSCC] section 2.3.9.

OutputBufferSize: The maximum number of bytes to return in **OutputBuffer**.

On completion, the object store MUST return:

Status: An NTSTATUS code that specifies the result.

OutputBuffer: An array of bytes that contains an 8-byte aligned array of **FILE_NAME_INFORMATION** ([MS-FSCC] section 2.1.7) structures. For more information, see [MS-FSCC] section 2.3.10.

BytesReturned: The number of bytes written to **OutputBuffer**.

This operation also uses the following local variables:

A list of **Links** (initialized to empty): *MatchingLinks*.

Unicode string: *RelativeName*.

32-bit unsigned integers (initialized to zero): *OutputBufferOffset*, *NameLength*.

Support for this operation is optional. If the object store does not implement this functionality, the operation MUST be failed with STATUS_INVALID_DEVICE_REQUEST.<55>

Pseudocode for the operation is as follows:

If **Open.Stream.StreamType** is DataStream, the operation MUST be failed with STATUS_INVALID_PARAMETER.

If **Open.HasManageVolumeAccess** is FALSE and **Open.HasBackupAccess** is FALSE, the operation MUST be failed with STATUS_ACCESS_DENIED.

If **Open.File.Volume.QuotaInformation** is empty, the operation MUST succeed with **BytesReturned** set to zero and **Status** set to STATUS_NO_QUOTAS_FOR_ACCOUNT.

If **OutputBufferSize** is less than 8, the minimum size required to return a **FILE_NAME_INFORMATION** structure with trailing padding, the operation MUST be failed with STATUS_INVALID_USER_BUFFER.

If **FindBySidData.Restart** is TRUE, **Open.FindBySidRestartIndex** MUST be set to zero.

For each *File* in **FindAllFiles**(**Open.File.Volume.RootDirectory**)<56>

If *File.***SecurityDescriptor.OwnerSid** matches **FindBySidData.SID** and *File.***FileNumber** is greater than or equal to **Open.FindBySidRestartIndex**, insert the first element of *File.***LinkList** into *MatchingLinks*.

EndFor

Sort *MatchingLinks* in ascending order by **File.FileNumber**.

For each *Link* in *MatchingLinks*:

Set *RelativeName* to ***BuildRelativeName***(*Link.***File**, **Open.File**).

If *RelativeName* is not empty (which means that *Link* represents **Open.File** or a descendant of it):

Strip off the leading backslash ("\") character from *RelativeName*.

Set *NameLength* to the length of *RelativeName*, in bytes.

If (**OutputBufferLength -** *OutputBufferOffset*) is less than **BlockAlign**(*NameLength* **+ 6, 8**):

BytesReturned is set to *OutputBufferOffset*.

If *OutputBufferOffset* is not zero:

The operation returns with STATUS_SUCCESS.

Else:

The operation MUST be failed with STATUS_BUFFER_TOO_SMALL.

EndIf

EndIf

Construct a **FILE_NAME_INFORMATION** structure starting at **OutputBuffer**[*OutputBufferOffset*], with the first 4 bytes (the **FileNameLength**) set to *NameLength*, and the next *NameLength* bytes (the **FileName**) set to *RelativeName*.

OutputBufferOffset = *OutputBufferOffset* + **BlockAlign**(*NameLength* **+ 6, 8**).

EndIf

Set **Open.FindBySidRestartIndex** to *Link***.File.FileNumber** + 1.

EndFor

Upon successful completion of the operation, the object store MUST return:

BytesReturned set to *OutputBufferOffset*.

Status set to STATUS_SUCCESS.

3.1.5.9.7 FSCTL_GET_COMPRESSION

Note: Some of the information in this section is subject to change because it applies to a preliminary implementation of the protocol or structure. For information about specific differences between versions, see the behavior notes that are provided in the Product Behavior appendix.

The server provides:

Open: An **Open** of a DataStream or DirectoryStream.

OutputBufferSize: The maximum number of bytes to return in **OutputBuffer**.

On completion, the object store MUST return:

Status: An NTSTATUS code that specifies the result.

OutputBuffer: An array of bytes that will return a USHORT value representing the compression state of the stream, as specified in [MS-FSCC] section 2.3.12.

BytesReturned: The number of bytes returned in **OutputBuffer**.

Support for this operation is optional. If the object store does not implement this functionality, the operation MUST be failed with STATUS_INVALID_DEVICE_REQUEST.<57>

Pseudocode for the operation is as follows:

If **OutputBufferSize** is less than ***sizeof(***USHORT***)*** (2 bytes), the operation MUST be failed with STATUS_INVALID_PARAMETER.

If **Open.Stream.StreamType** is DirectoryStream:

If **Open.File.FileAttributes**.FILE_ATTRIBUTE_COMPRESSED is TRUE:

The object store MUST set **OutputBuffer.CompressionState** to COMPRESSION_FORMAT_LZNT1.

Else:

The object store MUST set **OutputBuffer.CompressionState** to COMPRESSION_FORMAT_NONE.

EndIf

Else:

If **Open.Stream.IsCompressed** is TRUE:

The object store MUST set **OutputBuffer.CompressionState** to COMPRESSION_FORMAT_LZNT1.

Else:

The object store MUST set **OutputBuffer.CompressionState** to COMPRESSION_FORMAT_NONE.

EndIf

EndIf

Upon successful completion of the operation, the object store MUST return:

BytesReturned set to ***sizeof(***USHORT***)*** (2 bytes).

Status set to STATUS_SUCCESS.

3.1.5.9.8 FSCTL_GET_INTEGRITY_INFORMATION

Note: All of the information in this section is subject to change because it applies to a preliminary implementation of the protocol or structure.

The server provides:

Open: An **Open** of a DataStream or DirectoryStream.

OutputBufferSize: The maximum number of bytes to return in **OutputBuffer**.

Upon completion, the object store MUST return:

Status: An NTSTATUS code that specifies the result.

OutputBuffer: An array of bytes that will return an FSCTL_GET_INTEGRITY_INFORMATION_BUFFER structure, as specified in [MS-FSCC] section 2.3.46.

BytesReturned: The number of bytes returned in **OutputBuffer**.

Support for this operation is optional. If the object store does not implement this functionality, the operation MUST be failed with STATUS_INVALID_DEVICE_REQUEST.<58>

The operation MUST be failed with STATUS_INVALID_PARAMETER under any of the following conditions:

OutputBufferSize is less than ***sizeof(***FSCTL_GET_INTEGRITY_INFORMATION_BUFFER***)***.

Open.Stream.StreamType is not DirectoryStream or FileStream.

Open.File.FileAttributes.FILE_ATTRIBUTE_SYSTEM is TRUE.

Pseudocode for the operation is as follows:

The object store MUST initialize all fields in **OutputBuffer** to zero.

The object store MUST set **OutputBuffer.CheckSumAlgorithm** to one of the values for **ChecksumAlgorithm**, as specified in [MS-FSCC] section 2.3.46.

The object store MUST set **OutputBuffer.ChecksumChunkShift** to the base-2 logarithm of **Open.File.Volume.ChecksumChunkSize**.

The object store MUST set **OutputBuffer.ClusterShift** to the base-2 logarithm of **Open.File.Volume.ClusterSize**.

I6(-)1(2)-3(n□Tf1□2 2.546 16 Td[(O)-2(p)6(eu)3(s)5(seam)6(.l)-5(e)3(s)5(seam)6((_)-3(yp)3Ty)-13(p)6(e)]TJ/□T

If **OutputBufferSize** is less than ***sizeof(***NTFS_VOLUME_DATA_BUFFER***)***, the operation MUST be failed with STATUS_BUFFER_TOO_SMALL.

The object store MUST populate the fields of **OutputBuffer** as follows:<60>

OutputBuffer.VolumeSerialNumber set to **Open.File.Volume.VolumeSerialNumber**.

OutputBuffer.NumberSectors set to **Open.File.Volume.TotalSpace / Open.File.Volume.LogicalBytesPerSector**.

OutputBuffer.TotalClusters set to **Open.File.Volume.TotalSpace / Open.File.Volume.ClusterSize**.

OutputBuffer.FreeClusters set to **Open.File.Volume.FreeSpace / Open.File.Volume.ClusterSize**.

OutputBuffer.TotalReserved set to an implementation-specific value.

OutputBuffer.BytesPerSector set to **Open.File.Volume.LogicalBytesPerSector**.

OutputBuffer.BytesPerCluster set to **Open.File.Volume.ClusterSize**.

OutputBuffer.BytesPerFileRecordSegment set to an implementation-specific value.

OutputBuffer.ClustersPerFileRecordSegment set to an implementation-specific value.

OutputBuffer.MftValidDataLength set to an implementation-specific value.

OutputBuffer.MftStartLcn set to an implementation-specific value.

OutputBuffer.Mft2StartLcn set to an implementation-specific value.

OutputBuffer.MftZoneStart set to an implementation-specific value.

OutputBuffer.MftZoneEnd set to an implementation-specific value.

Upon successful completion of the operation, the object store MUST return:

BytesReturned set to ***sizeof(***NTFS_VOLUME_DATA_BUFFER***)***.

Status set to STATUS_SUCCESS.

3.1.5.9.10 FSCTL_GET_OBJECT_ID

The server provides:

Open: An **Open** of a DataFile or DirectoryFile.

OutputBufferSize: The maximum number of bytes to return in **OutputBuffer**.

On completion, the object store MUST return:

Status: An NTSTATUS code that specifies the result.

OutputBuffer: An array of bytes that will return a FILE_OBJECTID_BUFFER structure as specified in [MS-FSCC] section 2.1.3.

BytesReturned: The number of bytes returned in **OutputBuffer**.

Support for this operation is optional. If the object store does not implement this functionality, the operation MUST be failed with STATUS_INVALID_DEVICE_REQUEST.<61>

Pseudocode for the operation is as follows:

If **Open.File.Volume.IsObjectIDsSupported** is FALSE, the operation MUST be failed with STATUS_VOLUME_NOT_UPGRADED.

If **OutputBufferSize** is less than ***sizeof(***FILE_OBJECTID_BUFFER***)***, the operation MUST be failed with STATUS_INVALID_PARAMETER.

If **Open.File.ObjectId** is empty, the operation MUST be failed with STATUS_OBJECTID_NOT_FOUND.

The object store MUST populate the fields of **OutputBuffer** as follows:

OutputBuffer.ObjectId set to **Open.File.ObjectId**.

OutputBuffer.BirthVolumeId set to **Open.File.BirthVolumeId**.

OutputBuffer.BirthObjectId set to **Open.File.BirthObjectId**.

OutputBuffer.DomainId set to empty.

Upon successful completion of the operation, the object store MUST return:

BytesReturned set to ***sizeof (***FILE_OBJECTID_BUFFER***)***.

Status set to STATUS_SUCCESS.

3.1.5.9.11 FSCTL_GET_REPARSE_POINT

The server provides:

Open: An **Open** of a DataFile or DirectoryFile.

OutputBufferSize: The maximum number of bytes to return in **OutputBuffer**.

On completion, the object store **MUST** return:

OutputBuffer: An array of bytes containing a REPARSE_DATA_BUFFER or REPARSE_GUID_DATA_BUFFER structure as defined in [MS-FSCC] sections 2.1.2.2 and 2.1.2.3, respectively.

BytesReturned: The number of bytes returned to the caller.

Status: An NTSTATUS code that specifies the result.

Support for this operation is optional. If the object store does not implement this functionality, the operation MUST be failed with STATUS_INVALID_DEVICE_REQUEST.<62>

Pseudocode for the operation is as follows:

If **Open.File.Volume.IsReparsePointsSupported** is FALSE, the operation MUST be failed with STATUS_VOLUME_NOT_UPGRADED.

Phase 1 -- Check whether there is a reparse point on the **File**

If **Open.File.ReparseTag** is empty, the operation MUST be failed with STATUS_NOT_A_REPARSE_POINT.

Phase 2 -- Verify that **OutputBufferSize** is large enough to contain the reparse point data header.

If **Open.File.ReparseTag** is a Microsoft reparse tag as defined in [MS-FSCC] section 2.1.2.1, then **OutputBufferSize** MUST be >= ***sizeof(***REPARSE_DATA_BUFFER***)***. If not, the operation MUST be failed with STATUS_BUFFER_TOO_SMALL.

If **Open.File.ReparseTag** is a non-Microsoft reparse tag, then **OutputBufferSize** MUST be >= ***sizeof(***REPARSE_GUID_DATA_BUFFER***)***. If it is not, the operation MUST be failed with STATUS_BUFFER TOO_SMALL.

Phase 3 -- Return the reparse data

Set **OutputBuffer.ReparseTag** to **Open.File.ReparseTag**.

Set **OutputBuffer.ReparseDataLength** to the size of **Open.File.ReparseData**, in bytes.

Set **OutputBuffer.Reserved** to zero.

Copy as much of **Open.File.ReparseData** as can fit into the remainder of **OutputBuffer** starting at **OutputBuffer.DataBuffer**.

If **Open.File.ReparseTag** is a non-Microsoft reparse tag, set **OutputBuffer.ReparseGUID** to **Open.File.ReparseGUID**.

Upon successful completion of the operation, the object store MUST return:

BytesReturned set to the number of bytes written to **OutputBuffer**.

Status set to STATUS_SUCCESS.

3.1.5.9.12 FSCTL_GET_RETRIEVAL_POINTERS

The server provides:

Open: An **Open** of a DataStream or DirectoryStream.

StartingVcnBuffer: An array of bytes containing a STARTING_VCN_INPUT_BUFFER as described in [MS-FSCC] section 2.3.19.

OutputBufferSize: The maximum number of bytes to return in **OutputBuffer**.

On completion, the object store MUST return:

OutputBuffer: An array of bytes that will return a RETRIEVAL_POINTERS_BUFFER as defined in [MS-FSCC] section 2.3.20.

BytesReturned: The number of bytes returned to the caller.

Status: An NTSTATUS code that specifies the result.

Pseudocode for the operation is as follows:

Phase 1 -- Verify Parameters

If the size of **StartingVcnBuffer** is less than **sizeof** (STARTING_VCN_INPUT_BUFFER), the operation MUST be failed with STATUS_INVALID_PARAMETER.

If **OutputBufferSize** is smaller than ***sizeof(***RETRIEVAL_POINTERS_BUFFER***)***, the operation MUST be failed with STATUS_BUFFER_TOO_SMALL.

If **StartingVcnBuffer.StartingVcn** is negative, the operation MUST be failed with STATUS_INVALID_PARAMETER.

If **StartingVcnBuffer.StartingVcn** is greater than or equal to **Open.Stream.AllocationSize** divided by **Open.File.Volume.ClusterSize**, the operation MUST be failed with STATUS_END_OF_FILE.

Phase 2 -- Locate and copy the extents into **OutputBuffer**.

Find the first *Extent* in **Open.Stream.ExtentList** where *Extent***.NextVcn** is greater than **StartingVcnBuffer.StartingVcn**.

Set **OutputBuffer.StartingVcn** to the previous element's **NextVcn**. If the element is the first one in **Open.Stream.ExtentList**, set **OutputBuffer.StartVcn** to zero.

Copy as many EXTENTS elements from **Open.Stream.ExtentList** starting with *Extent* as will fit into the remaining space in **OutputBuffer**, at offset **OutputBuffer.Extents**.

Set **OutputBuffer.ExtentCount** to the number of EXTENTS elements copied.

Upon successful completion of the operation, the object store MUST return:

BytesReturned set to the number of bytes written to **OutputBuffer**.

Status set to STATUS_SUCCESS if all of the elements in **Open.Stream.ExtentList** were copied into **OutputBuffer.Extents**, else STATUS_BUFFER_OVERFLOW.

3.1.5.9.13 FSCTL_IS_PATHNAME_VALID

This operation always returns STATUS_SUCCESS.

3.1.5.9.14 FSCTL_LMR_GET_LINK_TRACKING_INFORMATION

This operation MUST be failed with STATUS_INVALID_DEVICE_REQUEST.

3.1.5.9.15 FSCTL_LMR_SET_LINK_TRACKING_INFORMATION

This operation MUST be failed with STATUS_INVALID_DEVICE_REQUEST.

3.1.5.9.16 FSCTL_OFFLOAD_READ

Note: All of the information in this section is subject to change because it applies to a preliminary implementation of the protocol or structure.

The server provides:

Open: An **Open** of a DataFile.

InputBuffer: An array of bytes containing a single FSCTL_OFFLOAD_READ_INPUT structure, as specified in [MS-FSCC] section 2.3.71, indicating the Token that indicates the range of the file to offload read, as specified in [MS-FSCC] section 2.3.73.

InputBufferSize: The number of bytes in **InputBuffer**.

OutputBufferSize: The number of bytes in **OutputBuffer**.

Upon completion, the object store MUST return:

Status: An NTSTATUS code that specifies the result.

OutputBuffer: An array of bytes that contains a single FSCTL_OFFLOAD_READ_OUTPUT structure, as specified in [MS-FSCC] section 2.3.72, which contains the Token for the read data, as specified in [MS-FSCC] section 2.3.73.

BytesReturned: The number of bytes written to **OutputBuffer**.

This operation also uses the following local variables:

Boolean (initialized to FALSE): *VdlSameAsEof*

32-bit unsigned integers (initialized to zero): *OutputBufferLength*

64-bit unsigned integers (initialized to zero): *StartingCluster*, *ValidDataLength*, *FileSize*, *LastClusterInFile*, *VdlTrimmedCopyLength*, and *StorageOffloadBytesRead*

A list of EXTENTS (initialized to empty): *OffloadLCNList*

An NTSTATUS code: *StorageOffloadReadStatus*

A STORAGE_OFFLOAD_TOKEN structure, as specified in [MS-FSCC] section 2.3.73: *StorageOffloadReadToken*

Support for this read operation is optional. If the object store does not implement this functionality, the operation MUST be failed with STATUS_INVALID_DEVICE_REQUEST.

Pseudocode for the operation is as follows:

If **Open.Volume.IsOffloadReadSupported** is FALSE, the operation MUST be failed with STATUS_NOT_SUPPORTED.

If **InputBufferSize** is less than the size of the FSCTL_OFFLOAD_READ_INPUT structure size, the operation MUST be failed with STATUS_BUFFER_TOO_SMALL.

If **OutputBufferSize** is less than the size of the FSCTL_OFFLOAD_READ_OUTPUT structure size, the operation MUST be failed with STATUS_BUFFER_TOO_SMALL.

If **InputBuffer.FileOffset** is NOT a multiple of **Open.Volume.BytesPerLogicalSector**, the operation MUST be failed with STATUS_INVALID_PARAMETER.

If **InputBuffer.Size** is not equal to the size of the FSCTL_OFFLOAD_READ_INPUT structure size, the operation MUST be failed with STATUS_INVALID_PARAMETER.

If the sum of **InputBuffer.FileOffset** and **InputBuffer.CopyLength** overflows 64 bits, the operation MUST be failed with STATUS_INVALID_PARAMETER.

If **InputBuffer.CopyLength** is equal to 0, the operation SHOULD return immediately with STATUS_SUCCESS.

If **Open.Stream.StreamType** != DataStream, the operation MUST be failed with STATUS_OFFLOAD_READ_FILE_NOT_SUPPORTED.

If **Open.Stream.IsSparse** is TRUE, the operation MUST be failed with STATUS_OFFLOAD_READ_FILE_NOT_SUPPORTED.

If **Open.Stream.IsEncrypted** is TRUE, the operation MUST be failed with STATUS_OFFLOAD_READ_FILE_NOT_SUPPORTED.

If **Open.Stream.IsCompressed** is TRUE, the operation MUST be failed with STATUS_OFFLOAD_READ_FILE_NOT_SUPPORTED.

If **Open.Stream.IsDeleted** is TRUE, the operation MUST be failed with STATUS_FILE_DELETED.

If **InputBuffer.FileOffset** / **Open.Volume.BytesPerCluster** is less than 0, the operation MUST be failed with STATUS_INVALID_PARAMETER.

Set *ValidDataLength* to **Open.Stream.ValidDataLength**.

Set *FileSize* to **Open.Stream.Size**.

If *ValidDataLength* is not equal to *FileSize*, set *VdlSameAsEof* to FALSE.

Set *StartingCluster* to InputBuffer.FileOffset / Open.Volume.BytesPerCluster.

Set *LastClusterInFile* to **ClustersFromBytesTruncate**(**Open.File.Volume**, *FileSize*).

If *StartingCluster* is greater than *LastClusterInFile*:

The operation MUST be failed with STATUS_END_OF_FILE.

Else If *StartingCluster* is less than 0:

The operation MUST be failed with STATUS_INVALID_PARAMETER.

EndIf

If **InputBuffer.FileOffset** is greater than or equal to *FileSize*, the operation MUST be failed with STATUS_END_OF_FILE.

If **InputBuffer.FileOffset** is greater than or equal to *ValidDataLength*:

Set **OutputBuffer.Token** to the Zero token as defined in [MS-FSCC] section 2.3.73.

The operation MUST return STATUS_SUCCESS, with **BytesReturned** set to **OutputBufferLength**, and **OutputBuffer.Flags** set to OFFLOAD_READ_FLAG_ALL_ZERO_BEYOND_CURRENT_RANGE.

EndIf

If the sum of **InputBuffer.FileOffset** and **InputBuffer.CopyLength** is greater than **ValidDataLength**:

Set **InputBuffer.CopyLength** to **ValidDataLength** –**InputBuffer.FileOffset**.

If *VdlSameAsEof* is TRUE:

Set **InputBuffer.CopyLength** to **BlockAlignTruncate**(**InputBuffer.CopyLength**, **Open.Volume.LogicalBytesPerSector**).

Set *VdlTrimmedCopyLength* to **InputBuffer.CopyLength**.

Set **OutputBuffer.Flags** to OFFLOAD_READ_FLAG_ALL_ZERO_BEYOND_CURRENT_RANGE.

EndIf

EndIf

For Each *Extent* in **Open.Stream.ExtentList** spanned by the range defined by **Input.FileOffset** and **Input.CopyLength**:

Append the partial or full *Extent* to *OffloadLCNList*.

EndFor

Construct the offload read command with the *OffloadLCNList* as the ranges, and *Token* length specified in **InputBuffer.CopyLength** as per [INCITS-T10/11-059] and send it to the underlying storage subsystem, storing the status from the operation in *StorageOffloadReadStatus*, the number of bytes represented by the token in *StorageOffloadBytesRead*, and the Token in *StorageOffloadToken*.

If the call was successful:

Set **OutputBuffer.Token** to *StorageOffloadToken*.

Set **OutputBuffer.TransferLength** to *StorageOffloadBytesRead*.

If **OutputBuffer.Flag** has the bit OFFLOAD_READ_FLAG_ALL_ZERO_BEYOND_CURRENT_RANGE set:

If **OutputBuffer.TransferLength** is less than *VdlTrimmedCopyLength*, clear the OFFLOAD_READ_FLAG_ALL_ZERO_BEYOND_CURRENT_RANGE bit in **OutputBuffer.Flags**.

EndIf

Else:

If *StorageOffloadReadStatus* is equal to STATUS_NOT_SUPPORTED or if *StorageOffloadReadStatus* is equal to STATUS_DEVICE_FEATURE_NOT_SUPPORTED, then set **Open.Volume.IsOffloadReadSupported** to FALSE.

EndIf

Upon successful completion of the operation, the object store MUST return:

BytesReturned set to **OutputBufferLength**.

Status set to STATUS_SUCCESS.

3.1.5.9.17 FSCTL_OFFLOAD_WRITE

Note: All of the information in this section is subject to change because it applies to a preliminary implementation of the protocol or structure.

The server provides:

Open: An **Open** of a DataFile.

InputBuffer: An array of bytes containing a single FSCTL_OFFLOAD_WRITE_INPUT structure, as specified in [MS-FSCC] section 2.3.74, indicating the Token to use as the source, and the range of the file to be offload written to, as specified in [MS-FSCC] section 2.3.73.

InputBufferSize: The number of bytes in **InputBuffer**.

OutputBufferSize: The number of bytes in **OutputBuffer**.

Upon completion, the object store MUST return:

Status: An NTSTATUS code that specifies the result.

OutputBuffer: An array of bytes that contains a single FSCTL_OFFLOAD_WRITE_OUTPUT structure, as specified in [MS-FSCC] section 2.3.75.

BytesReturned: The number of bytes written to **OutputBuffer**.

This operation also uses the following local variables:

32-bit unsigned integers (initialized to zero): *OutputBufferLength*

64-bit unsigned integers (initialized to zero): *NewValidDataLength*, *ValidDataLength*, *FileSize*, and *StorageOffloadBytesWritten*.

A list of EXTENTS (initialized to empty): *OffloadLCNList*

An NTSTATUS code: *StorageOffloadWriteStatus*

Support for this write operation is optional. If the object store does not implement this functionality, the operation MUST be failed with STATUS_INVALID_DEVICE_REQUEST.

Pseudocode for the operation is as follows:

If **Open.Volume.IsReadOnly** is TRUE, the operation MUST be failed with STATUS_MEDIA_WRITE_PROTECTED.

If **Open.Volume.IsOffloadWriteSupported** is FALSE, the operation MUST be failed with STATUS_NOT_SUPPORTED.

If **InputBufferSize** is less than the size of the **FSCTL_OFFLOAD_WRITE_INPUT** structure size, the operation MUST be failed with STATUS_BUFFER_TOO_SMALL.

If **OutputBufferSize** is less than the size of the **FSCTL_OFFLOAD_WRITE_OUTPUT** structure size, the operation MUST be failed with STATUS_BUFFER_TOO_SMALL.

If **InputBuffer.FileOffset** is NOT a multiple of **Open.Volume.BytesPerLogicalSector**, the operation MUST be failed with STATUS_INVALID_PARAMETER.

If **InputBuffer.CopyLength** is NOT a multiple of **Open.Volume.BytesPerLogicalSector**, the operation MUST be failed with STATUS_INVALID_PARAMETER.

If **InputBuffer.TransferOffset** is NOT a multiple of **Open.Volume.BytesPerLogicalSector**, the operation MUST be failed with STATUS_INVALID_PARAMETER.

If **InputBuffer.Size** is not equal to the size of the **FSCTL_OFFLOAD_WRITE_INPUT** structure size, the operation MUST be failed with STATUS_INVALID_PARAMETER.

If the sum of **InputBuffer.FileOffset** and **InputBuffer.CopyLength** overflows 64 bits, the operation MUST be failed with STATUS_INVALID_PARAMETER.

If **InputBuffer.CopyLength** is equal to 0, the operation SHOULD return immediately with STATUS_SUCCESS.

If **Open.Stream.StreamType** != **DataStream**, the operation MUST be failed with STATUS_OFFLOAD_WRITE_FILE_NOT_SUPPORTED.

If **Open.Stream.IsSparse** is TRUE, the operation MUST be failed with STATUS_OFFLOAD_WRITE_FILE_NOT_SUPPORTED.

If **Open.Stream.IsEncrypted** is TRUE, the operation MUST be failed with STATUS_OFFLOAD_WRITE_FILE_NOT_SUPPORTED.

If **Open.Stream.IsCompressed** is TRUE, the operation MUST be failed with STATUS_OFFLOAD_WRITE_FILE_NOT_SUPPORTED.

If **Open.Stream.IsDeleted** is TRUE, the operation MUST be failed with STATUS_FILE_DELETED.

If **InputBuffer.FileOffset** / **Open.Volume.BytesPerCluster** is less than 0, the operation MUST be failed with STATUS_INVALID_PARAMETER.

If (**InputBuffer.FileOffset** + **InputBuffer.CopyLength**) is greater than **Open.Volume.MaxFileSize**, the operation MUST be failed with STATUS_INVALID_PARAMETER.

If **Open.Volume.IsUsnJournalActive** is TRUE, the object store MUST post a USN change as per section 3.1.4.11 with **File** equal to **File**, **Reason** equal to USN_REASON_DATA_OVERWRITE, and **FileName** equal to **Open.File.Name**.

Set *FileSize* to **Open.Stream.Size**.

Set *ValidDataLength* to **Open.Stream.ValidDataLength**.

If **InputBuffer.FileOffset** is greater than or equal to **Open.Stream.FileSize**, the operation MUST be failed with STATUS_END_OF_FILE.

If **InputBuffer.FileOffset** is greater than *ValidDataLength*, the operation MUST be failed with STATUS_BEYOND_VDL.

For Each *Extent* in **Open.Stream.ExtentList** spanned by the range defined by **InputBuffer.FileOffset** and **InputBuffer.CopyLength**:

Append the partial or full *Extent* to *OffloadLCNList*.

EndFor

Construct the offload write command with the *OffloadLCNList* as the ranges, Token from **InputBuffer.Token**, token offset from **InputBuffer.TransferOffset**, and write length from **InputBuffer.CopyLength** as defined in [INCITS-T10/11-059] and send it to the underlying storage subsystem. Store the status from the operation in *StorageOffloadWriteStatus*, and the number of bytes written in *StorageOffloadBytesWritten*.

If the operation was successful:

Set *NewValidDataLength* to **InputBuffer.FileOffset** + *StorageOffloadBytesWritten*.

If *NewValidDataLength* is greater than *ValidDataLength*:

Set **Open.Stream.VDL** to *NewValidDataLength*.

EndIf

Set **OutputBuffer.LengthWritten** to *StorageOffloadBytesWritten*.

Set **OutputBuffer.Size** to the size of the FSCTL_OFFLOAD_WRITE_OUTPUT structure.

Set **OutputBuffer.Flags** to 0.

Else:

If *StorageOffloadWriteStatus* is equal to STATUS_NOT_SUPPORTED or if *OffloadWriteStatus* is equal to STATUS_DEVICE_FEATURE_NOT_SUPPORTED, then set **Open.Volume.IsOffloadWriteSupported** to FALSE.

EndIf

Upon successful completion of the operation, the object store MUST return:

BytesReturned set to *OutputBufferLength*.

Status set to STATUS_SUCCESS.

3.1.5.9.18 FSCTL_QUERY_FAT_BPB

Support for this operation is optional. If this operation is not supported, this operation MUST be failed with STATUS_INVALID_DEVICE_REQUEST.<63>

The server provides:

Open: An **Open** of a DataFile or DirectoryFile.

OutputBufferSize: The maximum number of bytes to return in **OutputBuffer**.

On completion, the object store MUST return:

Status: An NTSTATUS code that specifies the result.

OutputBuffer: An array of bytes that will return the first 0x24 bytes of sector zero, on a FAT volume.

BytesReturned: The number of bytes returned in **OutputBuffer**.

Support for this operation is optional. If the object store does not implement this functionality, the operation MUST be failed with STATUS_INVALID_DEVICE_REQUEST.<64>

Pseudocode for the operation is as follows:

If **OutputBufferSize** is less than 0x24, the operation MUST be failed with STATUS_BUFFER_TOO_SMALL.

The operation will now copy the first 0x24 bytes of sector 0 of the storage device associated with **Open.File.Volume** into **OutputBuffer**.

Upon successful completion of the operation, the object store MUST return:

BytesReturned set to 0x24.

Status set to STATUS_SUCCESS.

3.1.5.9.19 FSCTL_QUERY_ALLOCATED_RANGES

The server provides:

Open: An **Open** of a DataFile.

InputBuffer: An array of bytes containing a single FILE_ALLOCATED_RANGE_BUFFER structure indicating the range to query for allocation, as specified in [MS-FSCC] section 2.3.32.

InputBufferSize: The number of bytes in **InputBuffer**.

OutputBufferSize: The maximum number of bytes to return in **OutputBuffer**.

On completion, the object store MUST return:

Status: An NTSTATUS code that specifies the result.

OutputBuffer: An array of bytes that will return an array of zero or more FILE_ALLOCATED_RANGE_BUFFER structures as specified in [MS-FSCC] section 2.3.32.

BytesReturned: The number of bytes returned in **OutputBuffer**.

This operation uses the following local variables:

32-bit unsigned integer indicating the index of the next FILE_ALLOCATED_RANGE_BUFFER to fill in **OutputBuffer** (initialized to 0): *OutputBufferIndex*.

64-bit unsigned integer *QueryStart:* Is initialized to ***ClustersFromBytesTruncate***(**Open**.**File**.**Volume**, **InputBuffer**.**FileOffset**). This is the cluster containing the first byte of the queried range.

64-bit unsigned integer *QueryNext:* Is initialized to ***ClustersFromBytesTruncate***(**Open**.**File**.**Volume**, (**InputBuffer**.**FileOffset** + **InputBuffer**.**Length** - 1)) + 1. This is the cluster following the last cluster of the range.

64-bit unsigned integers (initialized to 0): *ExtentFirstVcn*, *ExtentNextVcn*, *RangeFirstVcn*, *RangeNextVcn*

Boolean values (initialized to FALSE): *FoundRangeStart*, *FoundRangeEnd*

Pointer to an EXTENTS element (initialized to NULL): *Extent*

FILE_ALLOCATED_RANGE_BUFFER (initialized to zeros)*: Range*

Support for this operation is optional. If the object store does not implement this functionality, the operation MUST be failed with STATUS_INVALID_DEVICE_REQUEST.<65>

Pseudocode for the operation is as follows:

If **Open**.**Stream.StreamType** is DirectoryStream, the operation MUST be failed with STATUS_INVALID_PARAMETER.

If **InputBufferSize** is less than ***sizeof(***FILE_ALLOCATED_RANGE_BUFFER***)***, the operation MUST be failed with STATUS_INVALID_PARAMETER.

If (**InputBuffer**.**FileOffset** < 0) or (**InputBuffer**.**Length** < 0) or (**InputBuffer**.**Length** > MAXLONGLONG - **InputBuffer**.**FileOffset**), the operation MUST be failed with STATUS_INVALID_PARAMETER. If **InputBuffer**.**Length** is 0:

Set **BytesReturned** to 0.

Return STATUS_SUCCESS.

EndIf

If **OutputBufferSize** < ***sizeof(***FILE_ALLOCATED_RANGE_BUFFER***)***, the operation MUST be failed with STATUS_BUFFER_TOO_SMALL.

If **Open**.**Stream**.**IsSparse** is FALSE:

Set **OutputBuffer**.**FileOffset** to **InputBuffer**.**FileOffset**.

Set **OutputBuffer**.**Length** to **InputBuffer**.**Length**.

Set **BytesReturned** to ***sizeof(***FILE_ALLOCATED_RANGE_BUFFER***)***.

Return STATUS_SUCCESS.

Else:

For sparse files, return a list of contiguous allocated ranges within the requested range. Contiguous allocated ranges in a sparse file might be fragmented on disk, therefore it is necessary to loop through the EXTENTS on this stream, coalescing the adjacent allocated EXTENTS into a single FILE_ALLOCATED_RANGE_BUFFER entry.

Set **Status** to STATUS_SUCCESS.

Set **BytesReturned** to 0.

For each *Extent* in **Open**.**Stream**.**ExtentList**:

Set *ExtentFirstVcn* to *ExtentNextVcn*.

Set *ExtentNextVcn* to *Extent*.**NextVcn**.

If *Extent*.**Lcn** != 0xffffffffffffffff, meaning *Extent* is allocated (not a sparse hole):

If *FoundRangeStart* is FALSE:

If *QueryStart* < *ExtentFirstVcn*:

Set *FoundRangeStart* to TRUE.

Set *RangeFirstVcn* to *ExtentFirstVcn*.

Else If *ExtentFirstVcn* <= *QueryStart* and *QueryStart* < *ExtentNextVcn*:

Set *FoundRangeStart* to TRUE.

Set *RangeFirstVcn* to *QueryStart*.

EndIf

EndIf

If *FoundRangeStart* is TRUE:

If *QueryNext* <= *ExtentFirstVcn*:

Break out of the For loop.

Else If *ExtentFirstVcn* < *QueryNext* and *QueryNext* <= *ExtentNextVcn*:

Set *FoundRangeEnd* to TRUE.

Set *RangeNextVcn* to *QueryNext*.

Else (*ExtentNextVcn* < *QueryNext*):

Set *FoundRangeEnd* to FALSE.

Set *RangeNextVcn* to *ExtentNextVcn*.

EndIf

EndIf

Else If *FoundRangeStart* is TRUE:

Set *FoundRangeEnd* to TRUE.

EndIf

If *FoundRangeEnd* is TRUE:

Set *FoundRangeStart* to FALSE and *FoundRangeEnd* to FALSE.

Add *Range* to *OutputBuffer* as follows:

Set *Range*.**FileOffset** to *RangeFirstVcn* * **Open**.**File**.**Volume**.**ClusterSize**.

Set *Range*.**Length** to (*RangeNextVcn* - *RangeFirstVcn*) * **Open**.**File**.**Volume**.**ClusterSize**.

If **OutputBufferSize** < ((*OutputBufferIndex* + 1) * ***sizeof(***FILE_ALLOCATED_RANGE_BUFFER***)***) then:

Set *RangeFirstVcn* to 0 and *RangeNextVcn* to 0.

Set **Status** to STATUS_BUFFER_OVERFLOW.

Break out of the For loop.

EndIf

Copy *Range* to **OutputBuffer**[*OutputBufferIndex*].

Increment *OutputBufferIndex* by 1.

Set *RangeFirstVcn* to 0 and *RangeNextVcn* to 0.

EndIf

EndFor

If *RangeNextVcn* is not 0:

If **OutputBufferSize** < ((*OutputBufferIndex* + 1) * ***sizeof(***FILE_ALLOCATED_RANGE_BUFFER***)***) then:

Set **Status** to STATUS_BUFFER_OVERFLOW.

Else add *Range* to *OutputBuffer* as follows:

Set *Range*.**FileOffset** to *RangeFirstVcn* * **Open.File.Volume.ClusterSize**.

Set *Range*.**Length** to (*RangeNextVcn* - *RangeFirstVcn*) * **Open.File.Volume.ClusterSize**.

Copy *Range* to **OutputBuffer**[*OutputBufferIndex*].

Increment *OutputBufferIndex* by 1.

EndIf

EndIf

Bias the first and the last returned ranges so that they match the offset/length passed in, using the following algorithm:

If *OutputBufferIndex* > 0:

If **OutputBuffer**[0]**.FileOffset** < **InputBuffer.FileOffset**:

Set **OutputBuffer**[0]**.Length** to **OutputBuffer**[0]**.Length** - (**InputBuffer.FileOffset** - **OutputBuffer**[0]**.FileOffset**).

Set **OutputBuffer**[0]**.FileOffset** to **InputBuffer.FileOffset**.

EndIf

If (**OutputBuffer**[*OutputBufferIndex* - *1*].**FileOffset** + **OutputBuffer**[*OutputBufferIndex* - *1*].**Length**) > (**InputBuffer**.**FileOffset** + **InputBuffer**.**Length**):

Set **OutputBuffer**[*OutputBufferIndex* - *1*]**.Length** to **InputBuffer.FileOffset** + **InputBuffer.Length** - **OutputBuffer**[*OutputBufferIndex* - *1*]**.FileOffset**.

EndIf

EndIf

Endif

Upon successful completion of the operation, the object store MUST return:

BytesReturned set to *OutputBufferIndex* * ***sizeof(***FILE_ALLOCATED_RANGE_BUFFER***)***.

Status set to STATUS_SUCCESS.

3.1.5.9.20 FSCTL_QUERY_ON_DISK_VOLUME_INFO

The server provides:

Open: An **Open** of a DataFile.

OutputBufferSize: The maximum number of bytes to return in **OutputBuffer**.

On completion, the object store MUST return:

Status: An NTSTATUS code that specifies the result.

OutputBuffer: An array of bytes that will return a FILE_QUERY_ON_DISK_VOL_INFO_BUFFER as defined in [MS-FSCC] section 2.3.36.

BytesReturned: The number of bytes returned in **OutputBuffer**.

Support for this operation is optional. If the object store does not implement this functionality, the operation MUST be failed with STATUS_INVALID_DEVICE_REQUEST.<66>

Pseudocode for the operation is as follows:

If **OutputBufferSize** is less than ***sizeof(***FILE_QUERY_ON_DISK_VOL_INFO_BUFFER***)***, the operation MUST be failed with STATUS_BUFFER_TOO_SMALL.

The object store MUST populate the fields of **OutputBuffer** as follows:

OutputBuffer.DirectoryCount set to **Open.File.Volume.DirectoryCount**.

OutputBuffer.FileCount set to **Open.File.Volume.FileCount**.

OutputBuffer.FsFormatMajVersion set to **Open.File.Volume.FsFormatMajVersion**.

OutputBuffer.FsFormatMinVersion set to **Open.File.Volume.FsFormatMinVersion**.

OutputBuffer.FsFormatName set to the Unicode string "UDF".

OutputBuffer.FormatTime set to **Open.File.Volume.FormatTime**.

OutputBuffer.LastUpdateTime set to **Open.File.Volume.LastUpdateTime**.

OutputBuffer.CopyrightInfo set to **Open.File.Volume.CopyrightInfo**.

OutputBuffer.AbstractInfo set to **Open.File.Volume.AbstractInfo**.

OutputBuffer.FormattingImplementationInfo set to **Open.File.Volume.FormattingImplementationInfo**.

OutputBuffer.LastModifyingImplementationInfo set to **Open.File.Volume.LastModifyingImplementationInfo**.

Upon successful completion of the operation, the object store MUST return:

BytesReturned set to ***sizeof(***FILE_QUERY_ON_DISK_VOL_INFO_BUFFER***)***.

Status set to STATUS_SUCCESS.

3.1.5.9.21 FSCTL_QUERY_SPARING_INFO

The server provides:

Open: An **Open** of a DataFile.

OutputBufferSize: The maximum number of bytes to return in **OutputBuffer**.

On completion, the object store MUST return:

Status: An NTSTATUS code that specifies the result.

OutputBuffer: An array of bytes that will return a FILE_QUERY_SPARING_BUFFER as defined in [MS-FSCC] section 2.3.38.

BytesReturned: The number of bytes returned in **OutputBuffer**.

Support for this operation is optional. If the object store does not implement this functionality, the operation MUST be failed with STATUS_INVALID_DEVICE_REQUEST.<67>

Pseudocode for the operation is as follows:

If **OutputBufferSize** is less than ***sizeof(***FILE_QUERY_SPARING_BUFFER***)***, the operation MUST be failed with STATUS_INVALID_PARAMETER.

The object store MUST populate the fields of **OutputBuffer** as follows:

OutputBuffer.SparingUnitBytes set to **Open.File.Volume.SparingUnitBytes**.

OutputBuffer.SoftwareSparing set to **Open.File.Volume.SoftwareSparing**.

OutputBuffer.TotalSpareBlocks set to **Open.File.Volume.TotalSpareBlocks**.

OutputBuffer.FreeSpareBlocks set to **Open.File.Volume.FreeSpareBlocks**.

Upon successful completion of the operation, the object store MUST return:

BytesReturned set to ***sizeof(:*** FILE_QUERY_SPARING_BUFFER***)***.

Status set to STATUS_SUCCESS.

3.1.5.9.22 FSCTL_READ_FILE_USN_DATA

The server provides:

Open: An **Open** of a DataFile or DirectoryFile.

OutputBufferSize: The maximum number of bytes to return in **OutputBuffer**.

On completion, the object store MUST return:

Status: An NTSTATUS code that specifies the result.

OutputBuffer: An array of bytes that will return a USN_RECORD as defined in [MS-FSCC] section 2.3.36.

BytesReturned: The number of bytes returned in **OutputBuffer**.

Support for this operation is optional. If the object store does not implement this functionality, the operation MUST be failed with STATUS_INVALID_DEVICE_REQUEST.<68>

This operation uses the following local variables:

Unicode string: *LinkNameToUse*

32-bit unsigned integers: *LinkNameLength*, *RecordLength*

Pseudocode for the operation is as follows:

If **OutputBufferSize** is less than ***sizeof(***USN_RECORD***)***, the operation MUST be failed with STATUS_BUFFER_TOO_SMALL.

The object store MUST choose a link name to use in constructing the reply, as shown in the following pseudocode:

```
Set LinkNameToUse to empty.
For each Link in Open.File.LinkList:
    If Link.ShortName is not empty:
        Set LinkNameToUse to Link.Name.
        Break out of the For loop.
    ElseIf LinkNameToUse is empty:
        Set LinkNameToUse to Link.Name.
    EndIf
EndFor
```

Set *LinkNameLength* to the length, in bytes, of *LinkNameToUse*.

Set *RecordLength* to ***BlockAlign(FieldOffset(***USN_RECORD.FileName***)*** + *LinkNameLength*, 8***)***.

If **OutputBufferSize** is less than *RecordLength*, the operation MUST be failed with STATUS_INFO_LENGTH_MISMATCH.

The object store MUST fill in the fields of **OutputBuffer** as follows:

OutputBuffer.RecordLength set to *RecordLength*.

OutputBuffer.MajorVersion set to 2.

OutputBuffer.MinorVersion set to 0.

OutputBuffer.FileReferenceNumber set to **Open.File.FileID**.

OutputBuffer.ParentFileReferenceNumber set to **Open.Link.ParentFile.FileID**.

OutputBuffer.Usn set to **Open.File.Usn**.

OutputBuffer.TimeStamp set to 0.

OutputBuffer.Reason set to 0.

OutputBuffer.SourceInfo set to 0.

OutputBuffer.SecurityId set to 0.

OutputBuffer.FileAttributes set to **Open.File.FileAttributes**, or to FILE_ATTRIBUTE_NORMAL if **Open.File.FileAttributes** is 0.

OutputBuffer.FileNameLength set to *RecordLength*.

OutputBuffer.FileName set to *LinkNameToUse*.

Padding bytes of zeroes to bring the total number of bytes written into **OutputBuffer** up to *RecordLength*.

Upon successful completion of the operation, the object store MUST return:

BytesReturned set to *RecordLength*.

Status set to STATUS_SUCCESS.

3.1.5.9.23 FSCTL_RECALL_FILE

The server provides:

Open: An **Open** of a DataFile.

On completion, the object store MUST return:

Status: An NTSTATUS code that specifies the result.

Support for this operation is optional. If the object store does not implement this functionality, the operation MUST be failed with STATUS_INVALID_DEVICE_REQUEST.<69>

Pseudocode for the operation is as follows:

```
If Open.File.FileType is DirectoryFile, the operation MUST be failed with
    STATUS_INVALID_HANDLE.

        If Open.File.FileAttributes.FILE_ATTRIBUTE_OFFLINE is not set:


            // The file has already been recalled.

        Else


            Recall Open.File from remote storage.

            Clear Open.File.FileAttributes.FILE_ATTRIBUTE_OFFLINE

        EndIf
```

Upon successful completion of the operation, the object store MUST return:

Status set to STATUS_SUCCESS.

3.1.5.9.24 FSCTL_SET_COMPRESSION

The server provides:

Open: An **Open** of a DataFile or DirectoryFile.

InputBuffer: An array of bytes containing a USHORT value indicating the requested compression state of the stream, as specified in [MS-FSCC] section 2.3.43.

InputBufferSize: The number of bytes in **InputBuffer**.

On completion, the object store MUST return:

Status: An NTSTATUS code that specifies the result.

Support for this operation is optional. If the object store does not implement this functionality, the operation MUST be failed with STATUS_INVALID_DEVICE_REQUEST.<70>

The operation MUST be failed with STATUS_INVALID_PARAMETER under any of the following conditions:

InputBufferSize is less than ***sizeof(***USHORT***)*** (2 bytes).

InputBuffer.CompressionState is not one of the predefined values in [MS-FSCC] section 2.3.47.

Pseudocode for the operation is as follows:

If **InputBuffer.CompressionState** != COMPRESSION_FORMAT_NONE:

If compression support is disabled in the object store,<71> the operation MUST be failed with STATUS_COMPRESSION_DISABLED.

If **Open.File.Volume.ClusterSize** is greater than 4,096, the operation MUST be failed with STATUS_INVALID_DEVICE_REQUEST, because compression is not supported on volumes with a cluster size greater than 4 KB.

EndIf

If **Open.File.Volume.IsReadOnly** is TRUE, the operation MUST be failed with STATUS_MEDIA_WRITE_PROTECTED.

If **Open.Stream.IsEncrypted** is TRUE, the operation MUST be failed with STATUS_INVALID_DEVICE_REQUEST.

If (**InputBuffer.CompressionState** == COMPRESSION_FORMAT_NONE and **Open.Stream.IsCompressed** is FALSE) or (**InputBuffer.CompressionState** != COMPRESSION_FORMAT_NONE and **Open.Stream.IsCompressed** is TRUE), the operation MUST return STATUS_SUCCESS at this point.

The object store MUST initialize *ChangedAllocation* to FALSE.

The object store MUST post a USN change as per section 3.1.4.11 with **File** equal to **File**, **Reason** equal to USN_REASON_COMPRESSION_CHANGE, and **FileName** equal to **Open.Link.Name**.

If **InputBuffer.CompressionState** != COMPRESSION_FORMAT_NONE:

If **Open.Stream.AllocationSize** is less than ***BlockAlign***(**Open.Stream.AllocationSize**, **Open.File.Volume.CompressionUnitSize**), the object store MUST increase **Open.Stream.AllocationSize** to ***BlockAlign***(**Open.Stream.AllocationSize**, **Open.File.Volume.CompressionUnitSize**). If there is not enough disk space, the operation MUST be failed with STATUS_DISK_FULL; otherwise the object store MUST set *ChangedAllocation* to TRUE.

EndIf

If **InputBuffer.CompressionState** == COMPRESSION_FORMAT_NONE, the object store MUST set **Open.Stream.IsCompressed** to FALSE; otherwise it MUST be set to TRUE.

If **Open.Stream.StreamType** is DirectoryStream or **Open.Stream.Name** is empty, the object store MUST propagate the compression state to **Open.File**:

If **Open.Stream.IsCompressed** is TRUE, the object store MUST set **Open.File.FileAttributes**.FILE_ATTRIBUTE_COMPRESSED to TRUE; otherwise it MUST be set to FALSE.

EndIf

Send directory change notification as per section 3.1.4.1, with **Volume** equal to **Open.File.Volume**, **Action** equal to FILE_ACTION_MODIFIED, **FilterMatch** equal to FILE_NOTIFY_CHANGE_ATTRIBUTES, and **FileName** equal to **Open.FileName**.

If **Open.Stream.StreamType** is DirectoryStream, the operation MUST return STATUS_SUCCESS at this point.

If **Open.Stream.IsCompressed** is FALSE and **Open.Stream.AllocationSize** is greater than ***BlockAlign***(**Open.Stream.Size**, **Open.File.Volume.ClusterSize**), the object store SHOULD free excess allocation by setting **Open.Stream.AllocationSize** to ***BlockAlign***(**Open.Stream.Size**, **Open.File.Volume.ClusterSize**). If any allocation is freed in this way, the object store MUST set *ChangedAllocation* to TRUE.

If **Open.Stream.IsSparse** is TRUE, the object store SHOULD free any allocated compression unit-aligned extents beyond **Open.Stream.ValidDataLength**. If any allocation is freed in this way, the object store MUST set *ChangedAllocation* to TRUE.

If *ChangedAllocation* is TRUE and **Open.Stream.Name** is empty, the object store MUST set **Open.File.PendingNotifications.**FILE_NOTIFY_CHANGE_SIZE to TRUE.

Upon successful completion of the operation, the object store MUST return:

Status set to STATUS_SUCCESS.

3.1.5.9.25 FSCTL_SET_DEFECT_MANAGEMENT

The server provides:

Open: An **Open** of a DataStream.

InputBuffer: An array of bytes containing a Boolean as specified in [MS-FSCC] section 2.3.49.

InputBufferSize: The number of bytes in **InputBuffer**.

On completion, the object store MUST return:

Status: An NTSTATUS code that specifies the result.

Support for this operation is optional. If the object store does not implement this functionality or the target media is not a software defect-managed media, the operation MUST be failed with STATUS_INVALID_DEVICE_REQUEST.<72>

Pseudocode for the operation is as follows:

If **Open.Stream.StreamType** is DirectoryStream, the operation MUST be failed with STATUS_INVALID_PARAMETER.

If **InputBufferSize** is less than ***sizeof(***Boolean***)*** (1 byte), the operation MUST be failed with STATUS_INVALID_PARAMETER.

If **Open.File.OpenList** contains more than one Open on this stream, this operation MUST be failed with STATUS_SHARING_VIOLATION.

The object store MUST set **Open.File.DisableDefectManagement** to **InputBuffer.Disable**.

Upon successful completion of the operation, the object store MUST return:

Status set to STATUS_SUCCESS.

3.1.5.9.26 FSCTL_SET_ENCRYPTION

Note: Some of the information in this section is subject to change because it applies to a preliminary implementation of the protocol or structure. For information about specific differences between versions, see the behavior notes that are provided in the Product Behavior appendix.

The server provides:

Open: An **Open** of a DataFile or DirectoryFile.

InputBuffer: An array of bytes containing an ENCRYPTION_BUFFER structure indicating the requested encryption state of the stream or file, as specified in [MS-FSCC] section 2.3.49.

InputBufferSize: The number of bytes in **InputBuffer**.

On completion, the object store MUST return:

Status: An NTSTATUS code that specifies the result.

This operation uses the following local variables:

Boolean value (initialized to FALSE): *ChangedFileEncryption*

Support for this operation is optional. If the object store does not implement this functionality, the operation MUST be failed with STATUS_INVALID_DEVICE_REQUEST.<73>

Pseudocode for the operation is as follows:

If **Open.File.Volume.IsReadOnly** is TRUE, the operation MUST be failed with STATUS_MEDIA_WRITE_PROTECTED.

If **InputBufferSize** is smaller than ***BlockAlign(sizeof(***ENCRYPTION_BUFFER***)***, 4***)***, the operation MUST be failed with STATUS_BUFFER_TOO_SMALL.

The operation MUST be failed with STATUS_INVALID_PARAMETER under any of the following conditions:

If **InputBuffer.EncryptionOperation** is not one of the predefined values in [MS-FSCC] section 2.3.49.

If **InputBuffer.EncryptionOperation** == STREAM_SET_ENCRYPTION and **Open.Stream.IsCompressed** is TRUE.

If **InputBuffer.EncryptionOperation** == FILE_SET_ENCRYPTION:

If **Open.File.Attributes**.FILE_ATTRIBUTE_ENCRYPTED is FALSE:

The object store MUST set **Open.File.FileAttributes**.FILE_ATTRIBUTE_ENCRYPTED to TRUE.

The object store MUST set **Open.File.PendingNotifications**.FILE_NOTIFY_CHANGE_ATTRIBUTES to TRUE.

The object store MUST set *ChangedFileEncryption* to TRUE.

EndIf

ElseIf **InputBuffer.EncryptionOperation** == FILE_CLEAR_ENCRYPTION:

If **Open.File.Attributes**.FILE_ATTRIBUTE_ENCRYPTED is TRUE:

If there exists an *ExistingStream* in **Open.File.StreamList** such that *ExistingStream*.**IsEncrypted** is TRUE, the operation MUST be failed with STATUS_INVALID_DEVICE_REQUEST.

The object store MUST set **Open.File.FileAttributes**.FILE_ATTRIBUTE_ENCRYPTED to FALSE.

The object store MUST set **Open.File.PendingNotifications**.FILE_NOTIFY_CHANGE_ATTRIBUTES to TRUE.

The object store MUST set *ChangedFileEncryption* to TRUE.

EndIf

ElseIf **InputBuffer.EncryptionOperation** == STREAM_SET_ENCRYPTION:

If **Open.Stream.IsEncrypted** is FALSE:

The object store MUST set **Open.Stream.IsEncrypted** to TRUE.

If **Open.File.Attributes**.FILE_ATTRIBUTE_ENCRYPTED is FALSE:

The object store MUST set **Open.File.FileAttributes**.FILE_ATTRIBUTE_ENCRYPTED to TRUE.

The object store MUST set **Open.File.PendingNotifications**.FILE_NOTIFY_CHANGE_ATTRIBUTES to TRUE.

EndIf

EndIf

Else: // **InputBuffer.EncryptionOperation** == STREAM_CLEAR_ENCRYPTION

If **Open.Stream.IsEncrypted** is TRUE:

The object store MUST set **Open.Stream.IsEncrypted** to FALSE.

If there does not exist an *ExistingStream* in **Open.File.StreamList** such that *ExistingStream*.**IsEncrypted** is TRUE:

The object store MUST set **Open.File.FileAttributes**.FILE_ATTRIBUTE_ENCRYPTED to FALSE.

The object store MUST set **Open.File.PendingNotifications**.FILE_NOTIFY_CHANGE_ATTRIBUTES to TRUE.

EndIf

EndIf

EndIf

If **Open.File.PendingNotifications** is nonzero:

Set *FilterMatch* = (**Open.File.PendingNotifications** | **Open.Link.PendingNotifications**).

Send directory change notification as per section 3.1.4.1, with **Volume** equal to **Open.File.Volume**, **Action** equal to FILE_ACTION_MODIFIED, **FilterMatch** equal to *FilterMatch*, and **FileName** equal to **Open.FileName**.

For each 7oae

3.1.5.9.27 FSCTL_SET_INTEGRITY_INFORMATION

Note: All of the information in this section is subject to change because it applies to a preliminary implementation of the protocol or structure.

The server provides:<74>

Open: An **Open** of a DataFile or DirectoryFile.

InputBuffer: An array of bytes containing an FSCTL_SET_INTEGRITY_INFORMATION_BUFFER structure indicating the requested integrity state of the directory or file, as specified in [MS-FSCC] section 2.3.51.

InputBufferSize: The number of bytes in **InputBuffer**.

On completion, the object store MUST return:

Status: An NTSTATUS code that specifies the result.

Support for this operation is optional. If the object store does not implement this functionality, the operation MUST be failed with STATUS_INVALID_DEVICE_REQUEST.

The operation MUST be failed with STATUS_INVALID_PARAMETER under any of the following conditions:

InputBufferSize is less than ***sizeof(***FILE_INTEGRITY_STREAM_INFORMATION***)***.

InputBuffer.ChecksumAlgorithm is not one of the predefined values in [MS-FSCC] section 2.3.51.

The operation is attempting to change the checksum state of a non-empty file; the integrity status of files can be changed only when they have not yet been written to.

Pseudocode for the operation is as follows:

If **Open.File.Volume.IsReadOnly** is TRUE, the operation MUST be failed with STATUS_MEDIA_WRITE_PROTECTED.

If **Open.Stream.StreamType** is DirectoryStream:

The object store MUST post a USN change as specified in section 3.1.4.11 with File equal to Directory, Reason equal to USN_REASON_INTEGRITY_CHANGE, and FileName equal to **Open.Link.Name**.

If **InputBuffer.ChecksumAlgorithm** != CHECKSUM_TYPE_UNCHANGED, the object store MUST set **Open.Stream.CheckSumAlgorithm** to **InputBuffer.ChecksumAlgorithm**.

EndIf

If **Open.Stream.StreamType** is FileStream:

The object store MUST post a USN change as specified in section 3.1.4.11 with File equal to File, Reason equal to USN_REASON_INTEGRITY_CHANGE, and FileName equal to **Open.Link.Name**.

If **InputBuffer.ChecksumAlgorithm** != CHECKSUM_TYPE_UNCHANGED, the object store MUST set **Open.Stream.CheckSumAlgorithm** to **InputBuffer.ChecksumAlgorithm**.

If **InputBuffer.Flags** == CHECKSUM_ENFORCEMENT_OFF, the object store MUST set **Open.Stream.StreamChecksumEnforcementOff** to TRUE.

EndIf

Upon successful completion of the operation, the object store MUST return:

Status set to STATUS_SUCCESS.

3.1.5.9.28 FSCTL_SET_OBJECT_ID

The server provides:

Open: An **Open** of a DataFile or DirectoryFile.

InputBuffer: An array of bytes containing a FILE_OBJECTID_BUFFER structure as specified in [MS-FSCC] section 2.1.3.

InputBufferSize: The number of bytes in **InputBuffer**.

On completion, the object store MUST return:

Status: An NTSTATUS code that specifies the result.

Support for this operation is optional. If the object store does not implement this functionality, the operation MUST be failed with STATUS_INVALID_DEVICE_REQUEST.<75>

Pseudocode for the operation is as follows:

If **InputBufferSize** is not equal to ***sizeof(***FILE_OBJECTID_BUFFER***)***, the operation MUST be failed with STATUS_INVALID_PARAMETER.

If **Volume.IsReadOnly** is TRUE, the operation MUST be failed with STATUS_MEDIA_WRITE_PROTECTED.

If **Open.File.Volume.IsObjectIDsSupported** is FALSE, the operation MUST be failed with STATUS_VOLUME_NOT_UPGRADED.

If **Open.HasRestoreAccess** is FALSE, the operation MUST be failed with STATUS_ACCESS_DENIED.

If **Open.File.ObjectId** is not empty, the operation MUST be failed with STATUS_OBJECT_NAME_COLLISION.

If **InputBuffer.ObjectId** is not unique on **Open.File.Volume**, the operation MUST be failed with STATUS_DUPLICATE_NAME.

Before completing the operation successfully, the object store MUST set:

Open.File.LastChangeTime to the current time.<76>

Post a USN change as per section 3.1.4.11 with **File** equal to **File**, **Reason** equal to USN_REASON_OBJECT_ID_CHANGE, and **FileName** equal to **Open.Link.Name**.

Open.File.ObjectId to **InputBuffer.ObjectId**.

Open.File.BirthVolumeId to **InputBuffer.BirthVolumeId**.

Open.File.BirthObjectId to **InputBuffer.BirthObjectId**.

Open.File.DomainId to **InputBuffer.DomainId**.

Upon successful completion of the operation, the object store MUST return:

Status set to STATUS_SUCCESS.

3.1.5.9.29 FSCTL_SET_OBJECT_ID_EXTENDED

The server provides:

Open: An **Open** of a DataFile or DirectoryFile.

InputBuffer: An array of bytes containing a FILE_OBJECTID_BUFFER structure as specified in [MS-FSCC] section 2.1.3.1.

InputBufferSize: The number of bytes in **InputBuffer**.

On completion, the object store MUST return:

Status: An NTSTATUS code that specifies the result.

Support for this operation is optional. If the object store does not implement this functionality, the operation MUST be failed with STATUS_INVALID_DEVICE_REQUEST.<77>

Pseudocode for the operation is as follows:

If **InputBufferSize** is not equal to ***sizeof(***ObjectId.ExtendedInfo***)*** (48 bytes), the operation MUST be failed with STATUS_INVALID_PARAMETER.

If **Volume.IsReadOnly** is TRUE, the operation MUST be failed with STATUS_MEDIA_WRITE_PROTECTED.

If **Open.File.Volume.IsObjectIDsSupported** is FALSE, the operation MUST be failed with STATUS_VOLUME_NOT_UPGRADED.

If **Open.GrantedAccess** contains neither FILE_WRITE_DATA nor FILE_WRITE_ATTRIBUTES, the operation MUST be failed with STATUS_ACCESS_DENIED.

If **Open.File.ObjectId** is empty, the operation MUST be failed with STATUS_OBJECTID_NOT_FOUND.

Before completing the operation successfully, the object store MUST set:

Open.File.LastChangeTime to the current time.<78>

Post a USN change as per section 3.1.4.11 with **File** equal to **File**, **Reason** equal to USN_REASON_OBJECT_ID_CHANGE, and **FileName** equal to **Open.Link.Name**.

Open.File.BirthVolumeId to **InputBuffer.BirthVolumeId**.

Open.File.BirthObjectId to **InputBuffer.BirthObjectId**.

Open.File.DomainId to **InputBuffer.DomainId**.

Upon successful completion of this operation, the object store MUST return:

Status set to STATUS_SUCCESS.

3.1.5.9.30 FSCTL_SET_REPARSE_POINT

The server provides:

Open: An **Open** of a DataFile or DirectoryFile.

InputBufferSize: The byte count of the **InputBuffer**.

InputBuffer: An array of bytes containing a REPARSE_DATA_BUFFER or REPARSE_GUID_DATA_BUFFER structure as defined in [MS-FSCC] sections 2.1.2.2 and 2.1.2.3, respectively.

On completion, the object store **MUST** return:

Status: An NTSTATUS code that specifies the result.

Support for this operation is optional. If the object store does not implement this functionality, the operation MUST be failed with STATUS_INVALID_DEVICE_REQUEST.<79>

Pseudocode for the operation is as follows:

Phase 1 -- Verify the parameters

If (**Open.GrantedAccess** & (FILE_WRITE_DATA | FILE_WRITE_ATTRIBUTES)) == 0, the operation MUST be failed with STATUS_ACCESS_DENIED.

If **Open.File.Volume.IsReadOnly** is TRUE, the operation MUST be failed with STATUS_MEDIA_WRITE_PROTECTED.

If **Open.File.Volume.IsReparsePointsSupported** is FALSE, the operation MUST be failed with STATUS_VOLUME_NOT_UPGRADED.

If **InputBufferSize** is smaller than 8 bytes, the operation MUST be failed with STATUS_IO_REPARSE_DATA_INVALID.

If **InputBufferSize** is larger than 16384 bytes, the operation MUST be failed with STATUS_IO_REPARSE_DATA_INVALID.

If (**InputBufferSize** != **InputBuffer.ReparseDataLength** + 8) && (**InputBufferSize** != **InputBuffer.ReparseDataLength** + 24), the operation MUST be failed with STATUS_IO_REPARSE_DATA_INVALID.

If **InputBuffer.ReparseTag** == IO_REPARSE_TAG_MOUNT_POINT and **Open.File.FileType** != DirectoryFile, the operation MUST be failed with STATUS_NOT_A_DIRECTORY.

If **InputBuffer.ReparseTag** == IO_REPARSE_TAG_SYMLINK and **Open.HasCreateSymbolicLinkAccess** is FALSE, the operation MUST be failed with STATUS_ACCESS_DENIED.

If **Open.File.FileType** == DirectoryFile and **Open.File.DirectoryList** is not empty, the operation MUST be failed with STATUS_DIRECTORY_NOT_EMPTY.

If Open.File.FileType == DataFile and **InputBuffer.ReparseTag** == IO_REPARSE_TAG_SYMLINK and **Open.Stream.Size** is nonzero, the operation MUST be failed with STATUS_IO_REPARSE_DATA_INVALID.

If **Open.File.FileAttributes.**FILE_ATTRIBUTE_REPARSE_POINT is not set and **Open.File.ExtendedAttributesLength** is nonzero, the operation MUST be failed with STATUS_EAS_NOT_SUPPORTED.

Phase 2 -- Update the File

If **Open.File.ReparseTag** is not empty (indicating that a reparse point is already assigned):

If **Open.File.ReparseTag** != **InputBuffer.ReparseTag**, the operation MUST be failed with STATUS_IO_REPARSE_TAG_MISMATCH.

If **Open.File.ReparseTag** is a non-Microsoft tag and **Open.File.ReparseGUID** is not equal to **InputBuffer.ReparseGUID**, the operation MUST be failed with STATUS_REPARSE_ATTRIBUTE_CONFLICT.

Copy **InputBuffer.DataBuffer** to **Open.File.ReparseData**.

Else

Set **Open.File.ReparseTag** to **InputBuffer.ReparseTag**.

If **InputBuffer.ReparseTag** is a non-Microsoft Tag, then set **Open.File.ReparseGUID** to **InputBuffer.ReparseGUID**.

Set **Open.File.ReparseData** to **InputBuffer.ReparseData**.

Set **Open.File.FileAttributes**.FILE_ATTRIBUTE_REPARSE_POINT to TRUE.

EndIf

If **Open.File.FileType** == DataFile, set **Open.File.FileAttributes**.FILE_ATTRIBUTE_ARCHIVE to TRUE.

Update **Open.File.LastChangeTime** to the current system time.<80>

Upon successful completion of the operation, the object store MUST return:

Status set to STATUS_SUCCESS.

3.1.5.9.31 FSCTL_SET_SHORT_NAME_BEHAVIOR

This control code is reserved for the **WinPE**<81>environment; the object store MUST return STATUS_INVALID_DEVICE_REQUEST.

3.1.5.9.32 FSCTL_SET_SPARSE

The server provides:

Open: An **Open** of a DataStream.

InputBufferSize: The byte count of the **InputBuffer**.

InputBuffer: A buffer of type FILE_SET_SPARSE_BUFFER as defined in [MS-FSCC] section 2.3.59.

On completion, the object store **MUST** return:

Status: An NTSTATUS code that specifies the result.

Support for this operation is optional. If the object store does not implement this functionality, the operation MUST be failed with STATUS_INVALID_DEVICE_REQUEST.<82>

Pseudocode for the operation is as follows:

If **Open.File.Volume.IsReadOnly** is TRUE, the object store MUST return STATUS_MEDIA_WRITE_PROTECTED.

If **Open.GrantedAccess.FILE_WRITE_DATA** is FALSE and **Open.GrantedAccess.FILE_WRITE_ATTRIBUTES** is FALSE, the operation MUST be failed with STATUS_ACCESS_DENIED.

The object store MUST post a USN change as per section 3.1.4.11 with **File** equal to **File**, **Reason** equal to USN_REASON_BASIC_INFO_CHANGE, and **FileName** equal to **Open.Link.Name**. If **InputBuffer.SetSparse** is TRUE:

The object store MUST set **Open.Stream.IsSparse** to TRUE.

The object store MUST set **Open.File.FileAttributes.**FILE_ATTRIBUTE_SPARSE_FILE to TRUE, indicating that at least one stream of the file is sparse.

Else

For each *Extent* in **Open.Stream.ExtentList**:

If *Extent*.LCN is un-allocated as per [MS-FSCC] 2.3.20.1:

The object store MUST fully allocate the *Extent*. If the space cannot be allocated, then the operation MUST be failed with STATUS_DISK_FULL. The object store is not required to revert any allocations performed during the operation.

EndIf

EndFor

The object store MUST set **Open.Stream.IsSparse** to FALSE.

If there does not exist an *ExistingStream* in **Open.File.StreamList** such that *ExistingStream*.**IsSparse** is TRUE:

The object store MUST set **Open.File.FileAttributes.**FILE_ATTRIBUTE_SPARSE_FILE to FALSE, indicating that no streams of the file are sparse.

EndIf

EndIf

Set **Open.File.PendingNotifications.**FILE_NOTIFY_CHANGE_ATTRIBUTES to TRUE.

Upon successful completion of this operation, the object store MUST return:

Status set to STATUS_SUCCESS.

3.1.5.9.33 FSCTL_SET_ZERO_DATA

Note: Some of the information in this section is subject to change because it applies to a preliminary implementation of the protocol or structure. For information about specific differences between versions, see the behavior notes that are provided in the Product Behavior appendix.

The server provides:

Open: An **Open** of a DataStream.

InputBufferSize: The byte count of the **InputBuffer**.

InputBuffer: An array of bytes containing a FILE_ZERO_DATA_INFORMATION structure as defined in [MS-FSCC] section 2.3.65.

On completion, the object store **MUST** return:

Status: An NTSTATUS code that specifies the result.

This algorithm uses the following local variables:

64-bit signed integers: *StartingOffset*, *CurrentBytes*, *CurrentOffset*, *CurrentFinalByte*, *NextVcn* *CurrentVcn*, *ClusterCount*

64-bit signed integer initialized to -1: *LastOffset*

Support for this operation is optional. If the object store does not implement this functionality, the operation MUST be failed with STATUS_INVALID_DEVICE_REQUEST.<83>

The operation MUST be failed with STATUS_INVALID_PARAMETER under any of the following conditions:

InputBufferSize is less than ***sizeof(***FILE_ZERO_DATA_INFORMATION***)***.

InputBuffer.FileOffset is less than 0.

InputBuffer.BeyondFinalZero is less than 0.

InputBuffer.FileOffset is greater than **InputBuffer.BeyondFinalZero**.

Open.Stream.StreamType is not DataStream.

Pseudocode for the operation is as follows:

If **Open.File.Volume.IsReadOnly** is TRUE, the operation MUST be failed with STATUS_MEDIA_WRITE_PROTECTED.

Set *StartingOffset* equal to **InputBuffer.FileOffset**.

While TRUE:

If **Open.Stream.IsDeleted** is TRUE, the operation MUST be failed with STATUS_FILE_DELETED.

If *StartingOffset* is greater than or equal to **Open.Stream.Size**, or if *StartingOffset* is greater than or equal to **InputBuffer.BeyondFinalZero**, break out of the while loop.

Set *CurrentBytes* to **InputBuffer.BeyondFinalZero** - *StartingOffset*.

If **InputBuffer.BeyondFinalZero** is greater than **Open.Stream.Size**, set *CurrentBytes* to **Open.Stream.Size** - *StartingOffset*.

If *CurrentBytes* is greater than 0x40000000 (1 gigabyte), set *CurrentBytes* to 0x40000000.

If **Open.Stream.Oplock** is not empty, the object store MUST check for an oplock break according to the algorithm in section 3.1.4.12, with input values as follows:

Open equal to this operation's **Open**

Oplock equal to **Open.Stream.Oplock**

Operation equal to "FS_CONTROL"

OpParams containing a member **ControlCode** containing "FSCTL_SET_ZERO_DATA"

The object store MUST check for byte range lock conflicts using the algorithm described in section 3.1.4.10 with **ByteOffset** set to *StartingOffset*, **Length** set to *CurrentBytes*, **IsExclusive** set to TRUE, **LockIntent** set to FALSE and **Open** set to **Open**. If a conflict is detected, the operation MUST be failed with STATUS_FILE_LOCK_CONFLICT.

The object store MUST post a USN change as per section 3.1.4.11 with **File** equal to **File**, **Reason** equal to USN_REASON_DATA_OVERWRITE, and **FileName** equal to **Open.Link.Name**.

The object store MUST note that the file has been modified as per section 3.1.4.17 with **Open** equal to **Open**.

If *LastOffset* is -1 and *StartingOffset* is greater than **Open.Stream.ValidDataLength**:

Zero the data in the file according to the algorithm in section 3.1.5.9.33.1, setting the algorithm's parameters as follows:

Pass in the current **Open**.

StartingZero equal to **Open.Stream.ValidDataLength**.

ByteCount equal to *StartingOffset* -**Open.Stream.ValidDataLength**.

EndIf

If **Open.Stream.IsCompressed** is TRUE, or if **Open.Stream.IsSparse** is TRUE:

Set *CurrentOffset* to *StartingOffset* & ~(**Open.File.Volume.CompressionUnitSize** - 1). This aligns the starting point to a compression unit boundary, since when setting zero ranges on a sparse or compressed file, allocation is deleted in compression unit-aligned chunks.

Set *CurrentFinalByte* to **InputBuffer.BeyondFinalZero**.

If *CurrentFinalByte* is greater than or equal to **Open.Stream.Size**, set *CurrentFinalByte* to ***BlockAlign***(**Open.Stream.Size**, **Open.File.Volume.CompressionUnitSize**).

Set *NextVcn* and *CurrentVcn* equal to ***ClustersFromBytesTruncate***(**Open.File.Volume**, *CurrentOffset*).

While an unallocated range of the file exists starting at *NextVcn*:

NextVcn += The size of the unallocated range in clusters.

If (*NextVcn* * **Open.File.Volume.ClusterSize**) is greater than or equal to *CurrentFinalByte*:

NextVcn = ***ClustersFromBytesTruncate***(**Open.File.Volume**, *CurrentFinalByte*).

Break out of the While loop.

EndIf

EndWhile

NextVcn = ***BlockAlignTruncate***(*NextVcn*, ***ClustersFromBytes***(**Open.File.Volume**, **Open.File.Volume.CompressionUnitSize**)). This aligns *NextVcn* to a compression unit boundary.

If *NextVcn* != *CurrentVcn*:

ClusterCount = *NextVcn* - *CurrentVcn*

CurrentVcn += *ClusterCount*

EndIf

CurrentOffset = (*CurrentVcn* * **Open.File.Volume.ClusterSize**)

If *CurrentOffset* >= *CurrentFinalByte*, break out of the while loop.

If *CurrentOffset* < *StartingOffset*:

If there are not enough free clusters on the storage media to accommodate a write of **Open.File.Volume.CompressionUnitSize** bytes, the operation MUST be failed with STATUS_DISK_FULL. The object store is not required to undo any file zeroing or range deallocation that has been performed during the operation.

CurrentBytes = **Open.File.Volume.CompressionUnitSize** - (*StartingOffset* - *CurrentOffset*)

If (*CurrentOffset* + **Open.File.Volume.CompressionUnitSize**) > *CurrentFinalByte*:

CurrentBytes = *CurrentFinalByte* - *StartingOffset*

EndIf

The object store MUST write *CurrentBytes* zeroes into the stream beginning at *CurrentOffset* + (*StartingOffset* & (**Open.File.Volume.CompressionUnitSize** - 1)).

CurrentOffset += (*StartingOffset* & (**Open.File.Volume.CompressionUnitSize** - 1))

ElseIf *CurrentOffset* + **Open.File.Volume.CompressionUnitSize** > *CurrentFinalByte*:

If there are not enough free clusters on the storage media to accommodate a write of **Open.File.Volume.CompressionUnitSize** bytes, the operation MUST be failed with STATUS_DISK_FULL. The object store is not required to undo any file zeroing or range deallocation that has been performed during the operation.

CurrentBytes = *CurrentFinalByte* & (**Open.File.Volume.CompressionUnitSize** - 1)

The object store MUST write *CurrentBytes* zeroes into the stream beginning at *CurrentOffset*.

Else

CurrentBytes = *CurrentFinalByte* - *CurrentOffset*

If *CurrentBytes* is greater than 0x40000000, set *CurrentBytes* to 0x40000000.

CurrentBytes = ***BlockAlignTruncate***(*CurrentBytes*, **Open.File.Volume.CompressionUnitSize**)

If (*CurrentBytes* != 0) and (*NextVcn* <= (*CurrentVcn* +***ClustersFromBytesTruncate***(**Open.File.Volume**, *CurrentBytes*) - 1)):

The object store MUST delete *CurrentVcn* + ***ClustersFromBytesTruncate***(**Open.File.Volume**, *CurrentBytes*) - 1 clusters of allocation from the stream starting with the cluster at *NextVcn*.

EndIf

EndIf

Else

CurrentOffset = *StartingOffset*

CurrentFinalByte = ((*CurrentOffset* + 0x40000) & -(0x40000))

If *CurrentFinalByte* is greater than or equal to **Open.Stream.Size**, set *CurrentFinalByte* to **Open.Stream.Size**.

If *CurrentFinalByte* is greater than **InputBuffer.BeyondFinalZero**, set *CurrentFinalByte* to **InputBuffer.BeyondFinalZero**.

CurrentBytes = *CurrentFinalByte* - *CurrentOffset*

If *CurrentBytes* != 0 and *CurrentOffset* is less than **Open.Stream.ValidDataLength**:

The object store MUST write *CurrentBytes* zeroes into the stream beginning at *CurrentOffset*.

EndIf

EndIf

If *CurrentOffset* + *CurrentBytes* is greater than **Open.Stream.ValidDataLength** and *StartingOffset* is less than **Open.Stream.ValidDataLength**:

The object store MUST set **Open.Stream.ValidDataLength** equal to *CurrentOffset* + *CurrentBytes*.

EndIf

LastOffset = *StartingOffset*

If *CurrentBytes* != 0, set *StartingOffset* equal to *CurrentOffset* + *CurrentBytes*.

EndWhile

If **Open.Mode** contains either FILE_NO_INTERMEDIATE_BUFFERING or FILE_WRITE_THROUGH, the object store MUST flush all changes to the stream made during this operation, including any file size changes, to stable storage, and MUST fail the operation if the underlying physical storage reports an error flushing the data.

Upon successful completion of the operation, the object store MUST return:

Status set to STATUS_SUCCESS.

3.1.5.9.33.1 Algorithm to Zero Data Beyond ValidDataLength

This algorithm returns no value.

The inputs for the algorithm are:

ThisOpen: The **Open** for the stream being zeroed.

StartingZero: A 64-bit signed integer. The offset into the stream to begin zeroing.

ByteCount: The number of bytes to zero.

The algorithm uses the following local variables:

64-bit signed integers: *ZeroStart*, *BeyondZeroEnd*, *LastCompressionUnit*, *ClustersToDeallocate*

Pseudocode for the algorithm is as follows:

Set *ZeroStart* to ***BlockAlign***(**StartingZero**, **ThisOpen.File.Volume.LogicalBytesPerSector**).

Set *BeyondZeroEnd* to ***BlockAlign***(**StartingZero** + **ByteCount**, **ThisOpen.File.Volume.LogicalBytesPerSector**).

If (**ThisOpen.Stream.IsCompressed** is FALSE) and (**ThisOpen.Stream.IsSparse** is FALSE) and (*ZeroStart* != **StartingZero**):

The object store MUST write zeroes into the stream from **StartingZero** to *ZeroStart*.

EndIf

If ((**ThisOpen.Stream.IsCompressed** is TRUE) or

(**ThisOpen.Stream.IsSparse** is TRUE)) and

(**ByteCount** > **ThisOpen.File.Volume.CompressionUnitSize** * 2):

If ***BlockAlign***(*ZeroStart*, **ThisOpen.File.Volume.CompressionUnitSize**) != *ZeroStart*:

The object store MUST write zeroes into the stream from *ZeroStart* to ***BlockAlign***(*ZeroStart*, **ThisOpen.File.Volume.CompressionUnitSize**).

The object store MUST set **ThisOpen.Stream.ValidDataLength** to ***BlockAlign***(*ZeroStart*, **ThisOpen.File.Volume.CompressionUnitSize**).

Set *ZeroStart* equal to ***BlockAlign***(*ZeroStart*, **ThisOpen.File.Volume.CompressionUnitSize**).

EndIf

Set *LastCompressionUnit* equal to ***BlockAlignTruncate***(*BeyondZeroEnd*, **ThisOpen.File.Volume.CompressionUnitSize**).

Set *ClustersToDeallocate* equal to ***ClustersFromBytes***(**ThisOpen.File.Volume**, *LastCompressionUnit* - *ZeroStart*).

The object store MUST delete *ClusterToDeallocate* clusters of allocation from the stream starting with the cluster at ***ClustersFromBytes***(**ThisOpen.File.Volume**, *ZeroStart*).

If *LastCompressionUnit* != *BeyondZeroEnd*:

The object store MUST write zeroes into the stream from *LastCompressionUnit* to *BeyondZeroEnd*.

The object store MUST set **ThisOpen.Stream.ValidDataLength** equal to **StartingZero** + **ByteCount**.

EndIf

The algorithm returns at this point.

EndIf

If *ZeroStart* = *BeyondZeroEnd*

The algorithm returns at this point.

EndIf

The object store MUST write zeroes into the stream from *ZeroStart* to *BeyondZeroEnd*.

The object store MUST set **ThisOpen.Stream.ValidDataLength** equal to **StartingZero** + **ByteCount**.

3.1.5.9.34 FSCTL_SET_ZERO_ON_DEALLOCATION

The server provides:

Open: An **Open** of a DataStream.

On completion the object store MUST return:

Status: An NTSTATUS code that specifies the result.

Support for this operation is optional. If the object store does not implement this functionality, the operation MUST be failed with STATUS_INVALID_DEVICE_REQUEST.<84>

The operation MUST be failed with STATUS_ACCESS_DENIED under either of the following conditions:

Open.Stream.StreamType is not DataStream.

Open.GrantedAccess contains neither FILE_WRITE_DATA nor FILE_APPEND_DATA.

Pseudocode for the operation is as follows:

The object store MUST set **Open.Stream.ZeroOnDeallocate** to TRUE.

Upon successful completion of the operation, the object store MUST return:

Status set to STATUS_SUCCESS.

3.1.5.9.35 FSCTL_SIS_COPYFILE

Note: Some of the information in this section is subject to change because it applies to a preliminary implementation of the protocol or structure. For information about specific differences between versions, see the behavior notes that are provided in the Product Behavior appendix.

The server provides:

Open: An **Open** of a DataStream or DirectoryStream.

InputBuffer: An array of bytes containing a single SI_COPYFILE structure indicating the source and destination files to copy, as specified in [MS-FSCC] section 2.3.65.

InputBufferSize: The number of bytes in **InputBuffer**.

On completion, the object store MUST return:

Status: An NTSTATUS code that specifies the result.

This routine uses the following local variables:

Opens: *SourceOpen*, *DestinationOpen*

The purpose of this operation is to make it look like a copy from the source file to the destination file has occurred when in reality no data is actually copied. This operation modifies the source file in such a way that the clusters associated with it can be shared across multiple files. The destination file is created and modified to point at the same shared clusters that the source file points to.<85>

Support for [SIS] is optional. If the object store does not implement this functionality, the operation MUST be failed with STATUS_INVALID_DEVICE_REQUEST.

Pseudocode for the operation is as follows:

If **Open.IsAdministrator** is FALSE, the operation MUST be failed with STATUS_ACCESS_DEFINED.

If **InputBufferSizes** is less than **sizeof(**SI_COPYFILE**)**, the operation MUST be failed with STATUS_INVALID_PARAMETER_1.

If **InputBuffer.Flags** contains any flags besides COPYFILE_SIS_LINK and COPYFILE_SIS_REPLACE, the operation MUST be failed with STATUS_INVALID_PARAMETER_2.

If **InputBuffer.SourceFileNameLength** or **InputBuffer.DestinationFileNameLength** is <= zero, the operation MUST be failed with STATUS_INVALID_PARAMETER_3.

If **InputBuffer.SourceFileNameLength** or **InputBuffer.DestinationFileNameLength** is > MAXUSHORT (0xffff), the operation MUST be failed with STATUS_INVALID_PARAMETER.

If ***FieldOffset(*InputBuffer.SourceFileName*)*** + **InputBuffer.SourceFileNameLength** + **InputBuffer.DestinationFileNameLength** is > **InputBufferSize**, the operation MUST be failed with STATUS_INVALID_PARAMETER_4.

SourceOpen set to the **Open** returned from a successful call to open a file as defined in section 3.1.5.1, setting the algorithm's parameters as follows:

RootOpen: Set to **Open.RootOpen**.

PathName: Set to **InputBuffer.SourceFileName**.

SecurityContext: Set to empty.<86>

DesiredAccess: Set to GENERIC_READ.

ShareAccess: If the source file is already controlled by SIS (meaning the source file already has a reparse point of type IO_REPARSE_TAG_SIS), then set to FILE_SHARE_READ, else set to zero.

CreateOptions: Set To FILE_NON_DIRECTORY_FILE | FILE_NO_INTERMEDIATE_BUFFERING.

CreateDisposition: Set to FILE_OPEN.

DesiredFileAttributes: Set to FILE_ATTRIBUTE_NORMAL.

IsCaseInsensitive: Set to TRUE.

TargetOplockKey: Set to Empty.

If the request fails, this operation MUST be failed with the returned STATUS.

The operation MUST be failed with STATUS_OBJECT_TYPE_MISMATCH under any of the following conditions:

If *SourceOpen*.**File.LinkList** contains more than one entry (meaning this file has hardlinks).

If *SourceOpen*.**Stream.IsEncrypted** is TRUE.

If *SourceOpen*.**File.ReparseTag** is empty or is not IO_REPARSE_TAG_SIS (as defined in [MS-FSCC] section 2.1.2.1) and **InputBuffer.Flags.COPYFILE_SIS_LINK** is TRUE.

If *SourceOpen*.**File.ReparseTag** is not empty and is not IO_REPARSE_TAG_SIS, the operation MUST be failed with STATUS_INVALID_PARAMETER.

DestinationOpen set to the **Open** returned from a successful call to create a file as defined in section 3.1.5.1, setting the algorithm's parameters as follows:

RootOpen: Set to **Open.RootOpen**.

PathName: Set to **InputBuffer.DestinationFileName**.

SecurityContext: Set to empty.<87>

DesiredAccess: Set to GENERIC_READ | GENERIC_WRITE | DELETE.

ShareAccess: Set to zero.

CreateOptions: Set to FILE_NON_DIRECTORY_FILE.

CreateDisposition: If **InputBuffer.Flags.COPYFILE_SIS_REPLACE** is TRUE, set to FILE_OVERWRITE_IF, else set to FILE_CREATE.

DesiredFileAttributes: Set to FILE_ATTRIBUTE_NORMAL.

IsCaseInsensitive: Set to TRUE.

TargetOplockKey: Set to Empty.

If the request fails, this operation MUST be failed with the returned STATUS.

If *SourceOpen*.**Volume** is not equal to *DestinationOpen*.**Volume** is not equal to **Open.Volume**, the operation MUST be failed with STATUS_NOT_SAME_DEVICE.

Share the clusters between the source and destination file.<88>

DestinationOpen.**ReparseTag** set to IO_REPARSE_TAG_SIS.

Upon successful completion of the operation, the object store MUST return:

Status set to STATUS_SUCCESS.

3.1.5.9.36 FSCTL_WRITE_USN_CLOSE_RECORD

The server provides:

Open: An **Open** of a DataStream or DirectoryStream.

OutputBufferSize: The maximum number of bytes to return in **OutputBuffer**.

On completion, the object store MUST return:

Status: An NTSTATUS code that specifies the result.

OutputBuffer: An array of bytes that will return a **Usn** structure representing the current USN of the file, as specified in [MS-FSCC] section 2.3.68.

BytesReturned: The number of bytes returned in **OutputBuffer**.

Support for this operation is optional. If the object store does not implement this functionality, the operation MUST be failed with STATUS_INVALID_DEVICE_REQUEST.<89>

Pseudocode for the operation is as follows:

If **Open.File.Volume.IsReadOnly** is TRUE, the operation MUST be failed with STATUS_MEDIA_WRITE_PROTECTED.

If **OutputBufferSize** is less than ***sizeof(*Usn*)***, the operation MUST be failed with STATUS_INVALID_PARAMETER.

If **Open.File.Volume.IsUsnJournalActive** is FALSE, the operation MUST be failed with STATUS_JOURNAL_NOT_ACTIVE.

The object store MUST post a USN change as per section 3.1.4.11 with **File** equal to **File**, **Reason** equal to USN_REASON_CLOSE, and **FileName** equal to **Open.Link.Name**.

The object store MUST populate the fields of **OutputBuffer** as follows:

OutputBuffer.Usn set to **Open.File.Usn**.

Upon successful completion of the operation, the object store MUST return:

BytesReturned set to ***sizeof(*Usn*)***.

Status set to STATUS_SUCCESS.

3.1.5.10 Server Requests Change Notifications for a Directory

The server provides:

Open: An **Open** of a DirectoryStream.

OutputBufferSize: The maximum number of bytes to return in **OutputBuffer**.

WatchTree: A Boolean indicating whether the directory should be monitored recursively.

CompletionFilter: A 32-bit unsigned integer composed of flags indicating the types of changes to monitor as specified in [MS-SMB2] section 2.2.35.

On completion, the object store MUST return:

Status: An NTSTATUS code that specifies the result.

OutputBuffer: An array of bytes containing the notification data.

ByteCount: The count of the bytes in the array.

Pseudocode for the operation is as follows:

The **Open.File.Volume.ChangeNotifyList** MUST be searched for a **ChangeNotifyEntry** where **ChangeNotifyEntry.OpenedDirectory** matches **Open**.

If there were no matching **ChangeNotifyEntries**, one MUST be constructed so that:

ChangeNotifyEntry.OpenedDirectory points to **Open**.

ChangeNotifyEntry.WatchTree is set to **WatchTree**.

ChangeNotifyEntry.CompletionFilter is set to **CompletionFilter**.

ChangeNotifyEntry.NotifyEventList is initialized to an empty list.

Insert **ChangeNotifyEntry** at the end of **Open.File.Volume.ChangeNotifyList**.

EndIf

Insert operation into **CancelableOperations.CancelableOperationList**.

Wait for a Change Notify per section 3.1.5.10.1

3.1.5.10.1 Waiting for Change Notification to be Reported

Wait until the following conditions are satisfied:

There are one or more elements in **ChangeNotifyEntry.NotifyEventList**.

This change notification request is the oldest outstanding request on this **Open**. This means multiple change notification requests on the same **Open** are completed sequentially and in first-in-first-out (FIFO) order.

The operation is canceled per section 3.1.5.19.

Pseudocode for the operation is as follows:

When a **ChangeNotifyEntry**.**NotifyEventList** element is available:

If all entries from ChangeNotifyEntry.NotifyEventList fit in OutputBufferSize bytes:

Remove all NotifyEventEntries from ChangeNotifyEntry.NotifyEventList.

Copy NotifyEventEntries to OutputBuffer.

Set Status to STATUS_SUCCESS.

Set ByteCount to the size of OutputBuffer, in bytes.

Else:

Set **Status** to STATUS_NOTIFY_ENUM_DIR.

Set **ByteCount** to zero.

EndIf

EndIf

3.1.5.11 Server Requests a Query of File Information

The server provides:

Open: An **Open** of a DataStream or DirectoryStream.

OutputBufferSize: The maximum number of bytes to be returned in **OutputBuffer**.

FileInformationClass: The type of information being queried, as specified in [MS-FSCC] section 2.4.

On completion, the object store MUST return:

Status: An NTSTATUS code that specifies the result.

OutputBuffer: An array of bytes containing the file information. The structure of these bytes is dependent on **FileInformationClass**, as noted in the relevant subsection.

ByteCount: The number of bytes stored in **OutputBuffer**.

If **FileInformationClass** is not defined in [MS-FSCC] section 2.4, the operation MUST be failed with STATUS_INVALID_INFO_CLASS.

3.1.5.11.1 FileAccessInformation

OutputBuffer is of type FILE_ACCESS_INFORMATION as described in [MS-FSCC] 2.4.1.

Pseudocode for the operation is as follows:

If **OutputBufferSize** is smaller than ***sizeof(***FILE_ACCESS_INFORMATION***)***, the operation MUST be failed with STATUS_INFO_LENGTH_MISMATCH.

OutputBuffer MUST be filled out as follows:

OutputBuffer.AccessFlags set to **Open.GrantedAccess**.

Upon successful completion of the operation, the object store MUST return:

ByteCount set to ***sizeof(***FILE_ACCESS_INFORMATION***)***

Status set to STATUS_SUCCESS.

3.1.5.11.2 FileAlignmentInformation

OutputBuffer of type FILE_ALIGNMENT_INFORMATION as described in [MS-FSCC] section 2.4.3.

Pseudocode for the operation is as follows:

If **OutputBufferSize** is smaller than ***sizeof(***FILE_ALIGNMENT_INFORMATION***)***, the operation MUST be failed with Status STATUS_INFO_LENGTH_MISMATCH.

OutputBuffer MUST be filled out as follows:

OutputBuffer.AlignmentRequirement set to one of the alignment requirement values specified in [MS-FSCC] section 2.4.3 based on the characteristics of the device on which the File is stored.

Upon successful completion of the operation, the object store MUST return:

ByteCount set to ***sizeof(***FILE_ALIGNMENT_INFORMATION***)***.

Status set to STATUS_SUCCESS.

3.1.5.11.3 FileAllInformation

OutputBuffer is of type FILE_ALL_INFORMATION as described in [MS-FSCC] 2.4.2.

Pseudocode for the operation is as follows:

If **OutputBufferSize** is smaller than ***BlockAlign(FieldOffset(***FILE_ALL_INFORMATION.NameInformation.FileName***)*** + 2, 8***)***, the operation MUST be failed with STATUS_INFO_LENGTH_MISMATCH.

The object store MUST populate the fields of **OutputBuffer** as follows:

OutputBuffer.BasicInformation MUST be filled using the algorithm described in section 3.1.5.11.6.

OutputBuffer.StandardInformation MUST be filled using the operation described in section 3.1.5.11.27.

OutputBuffer.InternalInformation MUST be filled using the operation described in section 3.1.5.11.17.

OutputBuffer.EaInformation MUST be filled using the operation described in section 3.1.5.11.10.

OutputBuffer.AccessInformation MUST be filled using the operation described in section 3.1.5.11.1.

OutputBuffer.PositionInformation MUST be filled using the operation described in section 3.1.5.11.23.

OutputBuffer.ModeInformation MUST be filled using the operation described in section 3.1.5.11.18.

OutputBuffer.AlignmentInformation MUST be filled using the operation described in section 3.1.5.11.2.

OutputBuffer.NameInformation MUST be filled using the operation described in section 3.1.5.11.19, saving the returned ByteCount in *NameInformationLength* and the returned Status in *NameInformationStatus*.

Upon successful completion of the operation, the object store MUST return:

ByteCount set to ***FieldOffset(***FILE_ALL_INFORMATION.NameInformation***)*** + *NameInformationLength*.

Status set to *NameInformationStatus*.

3.1.5.11.4 FileAlternateNameInformation

OutputBuffer is of type FILE_NAME_INFORMATION as described in [MS-FSCC] 2.4.5.

Pseudocode for the operation is as follows:

If **OutputBufferSize** is smaller than ***BlockAlign*(*FieldOffset(***FILE_NAME_INFORMATION.FileName***)*** + 2, 4***)***, the operation MUST be failed with STATUS_INFO_LENGTH_MISMATCH.

If **Open.Link.ShortName** is empty, the operation MUST be failed with STATUS_OBJECT_NAME_NOT_FOUND.

OutputBuffer MUST be filled out as follows:

OutputBuffer.FileNameLength set to the length, in bytes, of **Open.Link.ShortName**.

OutputBuffer.FileName set to **Open.Link.ShortName**.

Upon successful completion of the operation, the object store MUST return:

ByteCount set to ***FieldOffset(***FILE_NAME_INFORMATION.FileName***)*** + **OutputBuffer.FileNameLength**.

Status set to STATUS_SUCCESS.

3.1.5.11.5 FileAttributeTagInformation

Note: Some of the information in this section is subject to change because it applies to a preliminary implementation of the protocol or structure. For information about specific differences between versions, see the behavior notes that are provided in the Product Behavior appendix.

OutputBuffer is of type FILE_ATTRIBUTE_TAG_INFORMATION as defined in [MS-FSCC] section 2.4.6.

Pseudocode for the operation is as follows:

If **OutputBufferSize** is smaller than ***sizeof(***FILE_ATTRIBUTE_TAG_INFORMATION***)***, the operation MUST be failed with STATUS_INFO_LENGTH_MISMATCH.

If **Open.GrantedAccess** does not contain FILE_READ_ATTRIBUTES, the operation MUST be failed with STATUS_ACCESS_DENIED.

If **Open.Stream.StreamType** is DirectoryStream:

The object store MUST set **OutputBuffer.FileAttributes** equal to the value of **Open.File.FileAttributes**.

The object store MUST set FILE_ATTRIBUTE_DIRECTORY in **OutputBuffer.FileAttributes**.

Else:

This is a DataStream. The object store MUST set **OutputBuffer.FileAttributes** equal to the value of **Open.File.FileAttributes**. The following attribute values, if they are set in **Open.File.FileAttributes**, MUST NOT be copied to **OutputBuffer.FileAttributes** (attribute flags are defined in [MS-FSCC] section 2.6):

FILE_ATTRIBUTE_COMPRESSED

FILE_ATTRIBUTE_TEMPORARY

FILE_ATTRIBUTE_SPARSE_FILE

FILE_ATTRIBUTE_ENCRYPTED

FILE_ATTRIBUTE_INTEGRITY_STREAM<90>

If **Open.Stream.IsSparse** is TRUE, the object store MUST set FILE_ATTRIBUTE_SPARSE_FILE in **OutputBuffer.FileAttributes**.

If **Open.Stream.IsEncrypted** is TRUE, the object store MUST set FILE_ATTRIBUTE_ENCRYPTED in **OuputBuffer.FileAttributes**.

If **Open.Stream.IsTemporary** is TRUE, the object store MUST set FILE_ATTRIBUTE_TEMPORARY in **OutputBuffer.FileAttributes**.

If **Open.Stream.IsCompressed** is TRUE, the object store MUST set FILE_ATTRIBUTE_COMPRESSED in **OutputBuffer.FileAttributes**.

If **Open.Stream.IsIntegrity** is TRUE, the object store MUST set FILE_ATTRIBUTE_INTEGRITY_STREAM in **OutputBuffer.FileAttributes**.<91>

EndIf

If **OutputBuffer.FileAttributes** is 0, the object store MUST set FILE_ATTRIBUTE_NORMAL in **OutputBuffer.FileAttributes**.

OutputBuffer.ReparseTag MUST be set to **Open.File.ReparseTag**.

Upon successful completion of the operation, the object store MUST return:

ByteCount set to ***sizeof(***FILE_ATTRIBUTE_TAG_INFORMATION***)***.

Status set to STATUS_SUCCESS.

3.1.5.11.6 FileBasicInformation

Note: Some of the information in this section is subject to change because it applies to a preliminary implementation of the protocol or structure. For information about specific differences between versions, see the behavior notes that are provided in the Product Behavior appendix.

OutputBuffer is of type FILE_BASIC_INFORMATION as defined in [MS-FSCC] section 2.4.7.

Pseudocode for the operation is as follows:

If **OutputBufferSize** is smaller than ***BlockAlign*(*sizeof(*FILE_BASIC_INFORMATION*)*, 8*)*, the operation MUST be failed with STATUS_INFO_LENGTH_MISMATCH.

If **Open.GrantedAccess** does not contain FILE_READ_ATTRIBUTES, the operation MUST be failed with STATUS_ACCESS_DENIED.

The object store MUST set **OutputBuffer.CreationTime** equal to **Open.File.CreationTime**.

The object store MUST set **OutputBuffer.LastWriteTime** equal to **Open.File.LastModificationTime**.

The object store MUST set **OutputBuffer.ChangeTime** equal to **Open.File.LastChangeTime**.

The object store MUST set **OutputBuffer.LastAccessTime** equal to **Open.File.LastAccessTime**.

If **Open.Stream.StreamType** is DirectoryStream:

The object store MUST set **OutputBuffer.FileAttributes** equal to the value of **Open.File.FileAttributes**.

The object store MUST set FILE_ATTRIBUTE_DIRECTORY in **OutputBuffer.FileAttributes**.

Else:

This is a DataStream. The object store MUST set **OutputBuffer.FileAttributes** equal to the value of **Open.File.FileAttributes**. The following attribute values, if they are set in **Open.File.FileAttributes**, MUST NOT be copied to **OutputBuffer.FileAttributes** (attribute flags are defined in [MS-FSCC] section 2.6):

FILE_ATTRIBUTE_COMPRESSED

FILE_ATTRIBUTE_TEMPORARY

FILE_ATTRIBUTE_SPARSE_FILE

FILE_ATTRIBUTE_ENCRYPTED

FILE_ATTRIBUTE_INTEGRITY_STREAM<92>

If **Open.Stream.IsSparse** is TRUE, the object store MUST set FILE_ATTRIBUTE_SPARSE_FILE in **OutputBuffer.FileAttributes**.

If **Open.Stream.IsEncrypted** is TRUE, the object store MUST set FILE_ATTRIBUTE_ENCRYPTED in **OuputBuffer.FileAttributes**.

If **Open.Stream.IsTemporary** is TRUE, the object store MUST set FILE_ATTRIBUTE_TEMPORARY in **OutputBuffer.FileAttributes**.

If **Open.Stream.IsCompressed** is TRUE, the object store MUST set FILE_ATTRIBUTE_COMPRESSED in **OutputBuffer.FileAttributes**.

If **Open.Stream.IsIntegrity** is TRUE, the object store MUST set FILE_ATTRIBUTE_INTEGRITY_STREAM in **OutputBuffer.FileAttributes**.<93>

EndIf

If **OutputBuffer.FileAttributes** is 0, the object store MUST set FILE_ATTRIBUTE_NORMAL in **OutputBuffer.FileAttributes**.

Upon successful completion of the operation, the object store MUST return:

ByteCount set to ***sizeof(***FILE_BASIC_INFORMATION***)***.

Status set to STATUS_SUCCESS.

3.1.5.11.7 FileBothDirectoryInformation

This operation is not supported and MUST be failed with STATUS_ INVALID_INFO_CLASS.

3.1.5.11.8 FileCompressionInformation

OutputBuffer is of type FILE_COMPRESSION_INFORMATION as defined in [MS-FSCC] section 2.4.9.

Pseudocode for the operation is as follows:

If **OutputBufferSize** is smaller than ***sizeof(***FILE_COMPRESSION_INFORMATION***)***, the operation MUST be failed with STATUS_INFO_LENGTH_MISMATCH.

The object store MUST initialize all fields in **OutputBuffer** to zero.

If **Open.Stream.StreamType** is DirectoryStream:

If **Open.File.FileAttributes**.FILE_ATTRIBUTE_COMPRESSED is TRUE:

The object store MUST set **OutputBuffer.CompressionState** to COMPRESSION_FORMAT_LZNT1.

Else:

The object store MUST set **OutputBuffer.CompressionState** to COMPRESSION_FORMAT_NONE.

EndIf

Else:

The object store MUST set **OutputBuffer.CompressedFileSize** to the number of bytes actually allocated on the underlying physical storage for storing the compressed data. This value MUST be a multiple of **Open.File.Volume.ClusterSize** and MUST be less than or equal to **Open.Stream.AllocationSize**.

If **Open.Stream.IsCompressed** is TRUE:

The object store MUST set **OutputBuffer.CompressionState** to COMPRESSION_FORMAT_LZNT1.

Else:

The object store MUST set **OutputBuffer.CompressionState** to COMPRESSION_FORMAT_NONE.

EndIf

EndIf

If **OutputBuffer.CompressionState** is not equal to COMPRESSION_FORMAT_NONE, the object store MUST set:

OutputBuffer.CompressedUnitShift to the base-2 logarithm of **Open.File.Volume.CompressionUnitSize**.

OutputBuffer.ChunkShift to the base-2 logarithm of **Open.File.Volume.CompressedChunkSize**.

OutputBuffer.ClusterShift to the base-2 logarithm of **Open.File.Volume.ClusterSize**.

EndIf

Upon successful completion of the operation, the object store MUST return:

ByteCount set to ***sizeof(***FILE_COMPRESSION_INFORMATION***)***.

Status set to STATUS_SUCCESS.

3.1.5.11.9 FileDirectoryInformation

This operation is not supported and MUST be failed with STATUS_ INVALID_INFO_CLASS.

3.1.5.11.10 FileEaInformation

OutputBuffer is of type FILE_EA_INFORMATION as described in [MS-FSCC] 2.4.12.<94>

Pseudocode for the operation is as follows:

If **OutputBufferSize** is smaller than ***sizeof(***FILE_EA_INFORMATION***)***, the operation MUST be failed with STATUS_INFO_LENGTH_MISMATCH.

The object store MUST set:

OutputBuffer.EaSize set to **Open.File.ExtendedAttributesLength**. If **Open.File.ExtendedAttributesLength** is a nonzero value, **OutputBuffer.EaSize** is incremented by 4 to account for the header.

Upon successful completion of the operation, the object store MUST return:

ByteCount set to ***sizeof(***FILE_EA_INFORMATION***)***.

Status set to STATUS_SUCCESS.

3.1.5.11.11 FileFullDirectoryInformation

This operation is not supported and MUST be failed with STATUS_ INVALID_INFO_CLASS.

3.1.5.11.12 FileFullEaInformation

OutputBuffer is of type FILE_FULL_EA_INFORMATION as described in [MS-FSCC] 2.4.15.<95>

Pseudocode for the operation is as follows:

The object store MUST initialize **OutputBuffer** to zero.

If **Open.GrantedAccess** does not contain FILE_READ_EA, the operation MUST be failed with STATUS_ACCESS_DENIED.

If **Open.File.ExtendedAttributes** is not empty:

OutputBuffer is filled with as many complete FILE_FULL_EA_INFORMATION entries from **Open.File.ExtendedAttributes,** starting with **Open.NextEaEntry,** as can be contained in **OutputBufferSize** bytes.

Open.NextEaEntry is set to point to the entry after the last entry returned, if any.

Endif

Upon successful completion of the operation, the object store MUST return:

ByteCount set to the size, in bytes, of all FILE_FULL_EA_INFORMATION entries returned.

Status set to:

STATUS_NO_EAS_ON_FILE if there were no entries to return in **Open.File.ExtendedAttributes**.

STATUS_BUFFER_TOO_SMALL if **OutputBufferSize** is too small to hold **Open.NextEaEntry**. No entries are returned.

STATUS_BUFFER_OVERFLOW if at least one entry was returned in **OutputBuffer** but there are still additional entries to return.

STATUS_SUCCESS when one or more entries were returned from **Open.File.ExtendedAttributes** and there are no more entries to return.

3.1.5.11.13 FileHardLinkInformation

This operation is not supported and MUST be failed with STATUS_NOT_SUPPORTED.

3.1.5.11.14 FileIdBothDirectoryInformation

This operation is not supported and MUST be failed with STATUS_ INVALID_INFO_CLASS.

3.1.5.11.15 FileIdFullDirectoryInformation

This operation is not supported and MUST be failed with STATUS_ INVALID_INFO_CLASS.

3.1.5.11.16 FileIdGlobalTxDirectoryInformation

This operation is not supported and MUST be failed with STATUS_ INVALID_INFO_CLASS.

3.1.5.11.17 FileInternalInformation

OutputBuffer is of type FILE_INTERNAL_INFORMATION as described in [MS-FSCC] 2.4.20.

Pseudocode for the operation is as follows:

If **OutputBufferSize** is smaller than ***sizeof(***FILE_INTERNAL_INFORMATION***)***, the operation MUST be failed with STATUS_INFO_LENGTH_MISMATCH.

OutputBuffer MUST be filled out as follows:

OutputBuffer.IndexNumber set to **Open.File.FileID**.

Upon successful completion of the operation, the object store MUST return:

ByteCount set to ***sizeof(***FILE_INTERNAL_INFORMATION***)***.

Status set to STATUS_SUCCESS.

3.1.5.11.18 FileModeInformation

OutputBuffer is of type FILE_MODE_INFORMATION as described in [MS-FSCC] 2.4.24.

Pseudocode for the operation is as follows:

If **OutputBufferSize** is smaller than ***sizeof(***FILE_MODE_INFORMATION***)***, the operation MUST be failed with STATUS_INFO_LENGTH_MISMATCH.

OutputBuffer MUST be filled out as follows:

OutputBuffer.Mode MUST be set to **Open.Mode**.

Upon successful completion of the operation, the object store MUST return:

ByteCount set to ***sizeof(***FILE_MODE_INFORMATION***)***.

Status set to STATUS_SUCCESS.

3.1.5.11.19 FileNameInformation

This operation is not supported from a remote client, it is only supported from a local client or as part of processing a query for the FileAllInformation operation as specified in section 3.1.5.11.3. If used to query from a remote client, this operation MUST be failed with a status code of STATUS_NOT_SUPPORTED.

OutputBuffer is of type FILE_NAME_INFORMATION as described in [MS-FSCC] section 2.4.5.

This routine uses the following local variables:

Unicode string: *FileName*

32-bit unsigned integers: *FileNameLength*, *AvailableNameLength*

Pseudocode for the operation is as follows:

If **OutputBufferSize** is smaller than ***BlockAlign(FieldOffset(***FILE_NAME_INFORMATION.FileName***)*** + 2, 4***)***, the operation MUST be failed with a status code of STATUS_INFO_LENGTH_MISMATCH.

Set *FileName* to ***BuildRelativeName(*****Open.Link, Open.File.Volume.RootDirectory*****)***.

Set *FileNameLength* to the length, in bytes, of *FileName*.

Set **OutputBuffer.FileNameLength** to *FileNameLength*.

Set *AvailableNameLength* to ***BlockAlignTruncate((OutputBufferSize - FieldOffset(***FILE_NAME_INFORMATION.FileName***))***, 2***)***.

If *AvailableNameLength* < *FileNameLength*, the object store MUST fail the operation with:

AvailableNameLength bytes copied from *FileName* to **OutputBuffer.FileName**.

ByteCount set to ***FieldOffset(***FILE_NAME_INFORMATION.FileName***)*** + *AvailableNameLength*.

Status set to STATUS_BUFFER_OVERFLOW.

EndIf

Upon successful completion of the operation, the object store MUST return:

FileNameLength bytes copied from *FileName* to **OutputBuffer.FileName**.

ByteCount set to ***FieldOffset(***FILE_NAME_INFORMATION.FileName***)*** + *FileNameLength*.

Status set to STATUS_SUCCESS.

3.1.5.11.20 FileNamesInformation

This operation is not supported as a file information class, it is only supported as a directory information class, as specified in section 3.1.5.5.3.6. If used to query file information STATUS_INVALID_INFO_CLASS MUST be returned.

3.1.5.11.21 FileNetworkOpenInformation

Note: Some of the information in this section is subject to change because it applies to a preliminary implementation of the protocol or structure. For information about specific differences between versions, see the behavior notes that are provided in the Product Behavior appendix.

OutputBuffer is of type FILE_NETWORK_OPEN_INFORMATION as defined in [MS-FSCC] section 2.4.27.

Pseudocode for the operation is as follows:

If **OutputBufferSize** is smaller than ***sizeof(***FILE_NETWORK_OPEN_INFORMATION***)***, the operation MUST be failed with STATUS_INFO_LENGTH_MISMATCH.

If **Open.GrantedAccess** does not contain FILE_READ_ATTRIBUTES, the operation MUST be failed with STATUS_ACCESS_DENIED.

OutputBuffer MUST be filled out as follows:

OutputBuffer.CreationTime set to **Open.File.CreationTime**.

OutputBuffer.LastWriteTime set to **Open.File.LastModificationTime**.

OutputBuffer.ChangeTime set to **Open.File.LastChangeTime.**

OutputBuffer.LastAccessTime set to **Open.File.LastAccessTime**.

OutputBuffer.FileAttributes set to **Open.File.FileAttributes**.

If **Open.Stream.StreamType** is DirectoryStream:

FILE_ATTRIBUTE_DIRECTORY, as specified in [MS-FSCC] section 2.6, MUST always be set in **OutputBuffer.FileAttributes.**

Else:

For a DataStream, the following attribute values, as specified in [MS-FSCC] section 2.6, MUST NOT be copied to **OutputBuffer.FileAttributes**:

FILE_ATTRIBUTE_COMPRESSED

FILE_ATTRIBUTE_TEMPORARY

FILE_ATTRIBUTE_SPARSE_FILE

FILE_ATTRIBUTE_ENCRYPTED

FILE_ATTRIBUTE_INTEGRITY_STREAM<96>

If **Open.Stream.IsSparse** is TRUE, the object store MUST set FILE_ATTRIBUTE_SPARSE_FILE in **OutputBuffer.FileAttributes**.

If **Open.Stream.IsEncrypted** is TRUE, set FILE_ATTRIBUTE_ENCRYPTED in **OuputBuffer.FileAttributes**.

If **Open.Stream.IsTemporary** is TRUE, set FILE_ATTRIBUTE_TEMPORARY in **OutputBuffer.FileAttributes**.

If **Open.Stream.IsCompressed** is TRUE, set FILE_ATTRIBUTE_COMPRESSED in **OutputBuffer.FileAttributes**.

If **Open.Stream.IsIntegrity** is TRUE, the object store MUST set FILE_ATTRIBUTE_INTEGRITY_STREAM<97> in **OutputBuffer.FileAttributes**.

OutputBuffer.AllocationSize set to **Open.Stream.AllocationSize**.

OutputBuffer.EndOfFile set to **Open.Stream.Size**.

EndIf

If **OutputBuffer.FileAttributes** is 0, set FILE_ATTRIBUTE_NORMAL in **OutputBuffer.FileAttributes**.

Upon successful completion of the operation, the object store MUST return:

ByteCount set to ***sizeof(***FILE_NETWORK_OPEN_INFORMATION***)***.

Status set to STATUS_SUCCESS.

3.1.5.11.22 FileObjectIdInformation

This operation is not supported and MUST be failed with STATUS_NOT_SUPPORTED.

3.1.5.11.23 FilePositionInformation

OutputBuffer is of type FILE_POSITION_INFORMATION, as specified in [MS-FSCC] section 2.4.32.

Pseudocode for the operation is as follows:

If **OutputBufferSize** is less than the size, in bytes, of the FILE_POSITION_INFORMATION structure, the operation MUST be failed with STATUS_INFO_LENGTH_MISMATCH.

The objects store MUST set **OutputBuffer.CurrentByteOffset** equal to **Open.CurrentByteOffset**.

The operation returns STATUS_SUCCESS.<98>

3.1.5.11.24 FileQuotaInformation

This operation is not supported as a file information class; it is supported only as a server request, as specified in section 3.1.5.20. If used to query file information, STATUS_INVALID_PARAMETER MUST be returned.

3.1.5.11.25 FileReparsePointInformation

This operation is not supported as a file information class; it is only supported as a directory enumeration class, as specified in section 3.1.5.5.2. If used to query file information STATUS_NOT_SUPPORTED MUST be returned.

3.1.5.11.26 FileSfioReserveInformation

This operation is not supported and MUST be failed with STATUS_NOT_SUPPORTED.

3.1.5.11.27 FileStandardInformation

OutputBuffer is of type FILE_STANDARD_INFORMATION, as described in [MS-FSCC] 2.4.38.

Pseudocode for the operation is as follows:

If **OutputBufferSize** is smaller than ***sizeof(***FILE_STANDARD_INFORMATION***)***, the operation MUST be failed with STATUS_INFO_LENGTH_MISMATCH.

OutputBuffer MUST be filled out as follows:

If **Open.Stream.StreamType** is DirectoryStream, set **OutputBuffer.Directory** to 1 else 0.

If **Open.Stream.StreamType** is DirectoryStream or **Open.Stream.Name** is empty:

If **Open.Link.IsDeleted** is TRUE, set **OutputBuffer.DeletePending** to 1 else 0.

Else:

If **Open.Stream.IsDeleted** is TRUE, set **OutputBuffer.DeletePending** to 1 else 0.

EndIf

OutputBuffer.NumberOfLinks set to the number of **Link** elements in **Open.File.LinkList**, except if **Link.IsDeleted** field is TRUE (that is, the number of not-deleted links to the file).<99>

If **OutputBuffer.NumberOfLinks** is 0, set **OutputBuffer.DeletePending** to 1.

OutputBuffer.AllocationSize set to **Open.Stream.AllocationSize**.

OutputBuffer.EndOfFile set to **Open.Stream.Size**.

Upon successful completion of the operation, the object store MUST return:

ByteCount set to ***sizeof(***FILE_STANDARD_INFORMATION***)***.

Status set to STATUS_SUCCESS.

3.1.5.11.28 FileStandardLinkInformation

This operation is not supported and MUST be failed with STATUS_ INVALID_INFO_CLASS.

3.1.5.11.29 FileStreamInformation

OutputBuffer is of type FILE_STREAM_INFORMATION, as described in [MS-FSCC] 2.4.40.

This routine uses the following local variables:

32-bit unsigned integer: *StreamNameLength*, *RemainingLength*, *ThisElementSize*, *PreviousElementPadding*

Stream: *ThisStream*

Pointer to a buffer of type FILE_STREAM_INFORMATION: *CurrentPosition*, *LastPosition*

Pseudocode for the operation is as follows:

Initialize *PreviousElementPadding* to 0.

Initialize *CurrentPosition* to point to the 0th byte of **OutputBuffer**.

Initialize *RemainingLength* to be equal to **OutputBufferSize**.

For each **Stream** *ThisStream* of **Open.File**:

Set *StreamNameLength* equal to the length, in bytes, of *ThisStream*.**Name** plus the length, in bytes, of the Unicode string "$DATA" plus the length, in bytes, of two Unicode characters. This accommodates the length of the full stream name in the form :<*ThisStream*.**Name**>:$DATA.

Set *ThisElementSize* equal to the byte offset of *CurrentPosition*.**StreamName** plus *StreamNameLength*.

If *ThisElementSize* plus *PreviousElementPadding* is greater than *RemainingLength*, the operation MUST be failed with STATUS_BUFFER_OVERFLOW.

The object store MUST set *CurrentPosition*.**StreamSize** equal to *ThisStream*.**Size**.

The object store MUST set *CurrentPosition*.**AllocationSize** equal to *ThisStream*.**AllocationSize**.

The object store MUST set *CurrentPosition*.**StreamNameLength** equal to *StreamNameLength*.

The object store MUST set *CurrentPosition*.**StreamName** to the Unicode character ":", then append *ThisStream*.**Name**, then append the Unicode character ":", then append the Unicode string "$DATA".

Set *PreviousElementPadding* equal to **BlockAlign**(*ThisElementSize*, 8) minus *ThisElementSize*. The value *PreviousElementPadding* is used to align each FILE_STREAM_INFORMATION element in **OutputBuffer** on an 8-byte boundary.

The object store MUST set *CurrentPosition*.**NextEntryOffset** equal to *ThisElementSize* plus *PreviousElementPadding*.

Set *RemainingLength* equal to *RemainingLength* minus (*ThisElementSize* plus *PreviousElementPadding*).

Set *LastPosition* equal to *CurrentPosition*.

Advance *CurrentPosition* by a number of bytes equal to *ThisElementSize* plus *PreviousElementPadding*.

EndFor

The object store MUST set *LastPosition*.**NextEntryOffset** equal to 0.

The operation returns STATUS_SUCCESS.

3.1.5.12 Server Requests a Query of File System Information

The server provides:

Open: An **Open** of a DataFile or DirectoryFile.

OutputBufferSize: The maximum number of bytes to be returned in **OutputBuffer**.

FsInformationClass: The type of information being queried, as specified in [MS-FSCC] section 2.5.

On completion, the object store MUST return:

Status: An NTSTATUS code that specifies the result.

OutputBuffer: An array of bytes containing the file system information. The structure of these bytes is dependent on **FsInformationClass**, as noted in the relevant subsection.

ByteCount: The number of bytes stored in **OutputBuffer**.

Pseudocode for the operation is as follows:

If **FsInformationClass** is not defined in [MS-FSCC] section 2.5, the operation MUST be failed with STATUS_INVALID_PARAMETER.

3.1.5.12.1 FileFsVolumeInformation

OutputBuffer is of type FILE_FS_VOLUME_INFORMATION, as described in [MS-FSCC] 2.5.9.

This routine uses the following local variables:

32-bit unsigned integers: *RemainingLength*, *BytesToCopy*

Pseudocode for the operation is as follows:

If **OutputBufferSize** is smaller than ***BlockAlign(FieldOffset(***FILE_FS_VOLUME_INFORMATION.VolumeLabel***)***, 8***)***, the operation MUST be failed with STATUS_INFO_LENGTH_MISMATCH.

OutputBuffer MUST be filled out as follows:

OutputBuffer.VolumeCreationTime set to **Open.File.Volume.VolumeCreationTime**.

OutputBuffer.VolumeSerialNumber set to **Open.File.Volume.VolumeSerialNumber**.

OutputBuffer.VolumeLabelLength set to the length, in bytes, of the **Open.File.Volume.VolumeLabel** string. This value can be zero.

OutputBuffer.SupportsObjects set to TRUE.

Set *RemainingLength* to **OutputBufferSize -** ***FieldOffset(***FILE_FS_VOLUME_INFORMATION.VolumeLabel***)***.

If *RemainingLength* < **OutputBuffer.VolumeLabelLength**:

Set *BytesToCopy* to *RemainingLength*.

Else:

Set *BytesToCopy* to **OutputBuffer.VolumeLabelLength**.

EndIf

Copy *BytesToCopy* bytes from **Volume.VolumeLable** to **OutputBuffer.VolumeLabel**.

Upon successful completion of the operation, the object store MUST return:

ByteCount set to ***FieldOffset(***FILE_FS_VOLUME_INFORMATION.VolumeLabel***)*** + *BytesToCopy*.

Status set to STATUS_BUFFER_OVERFLOW if *BytesToCopy* < **OutputBuffer.VolumeLabelLength** else STATUS_SUCCESS.

3.1.5.12.2 FileFsLabelInformation

This operation is not supported and MUST be failed with STATUS_NOT_SUPPORTED.

3.1.5.12.3 FileFsSizeInformation

OutputBuffer is of type FILE_FS_SIZE_INFORMATION as described in [MS-FSCC] section 2.5.8.

This routine uses the following local variables:

64-bit unsigned integer: *RemainingQuota*

FILE_QUOTA_INFORMATION element: *QuotaEntry*

Pseudocode for the operation is as follows:

If **OutputBufferSize** is smaller than ***sizeof(***FILE_FS_SIZE_INFORMATION***)***, the operation MUST be failed with STATUS_INFO_LENGTH_MISMATCH.

OutputBuffer MUST be filled out as follows:

OutputBuffer.TotalAllocationUnits set to **Open.File.Volume.TotalSpace** / **Open.File.Volume.ClusterSize**.

OutputBuffer.AvailableAllocationUnits set to **Open.File.Volume.FreeSpace** / **Open.File.Volume.ClusterSize**.

OutputBuffer.SectorsPerAllocationUnit set to **Open.File.Volume.ClusterSize** / **Open.File.Volume.LogicalBytesPerSector**.

OutputBuffer.BytesPerSector set to **Open.File.Volume.LogicalBytesPerSector**.

If **Open.File.Volume.QuotaInformation** contains an entry *QuotaEntry* that matches the SID of the current **Open**, the object store MUST modify the returned information based on *QuotaEntry* as follows:

If *QuotaEntry*.**QuotaLimit** < **Open.File.Volume.TotalSpace**:

OutputBuffer.TotalAllocationUnits MUST be set to *QuotaEntry*.**QuotaLimit** / **Open.File.Volume.ClusterSize**.

EndIf

If *QuotaEntry*.**QuotaLimit** <= *QuotaEntry*.**QuotaUsed**:

RemainingQuota MUST be set to 0.

Else

RemainingQuota MUST be set to *QuotaEntry*.**QuotaLimit** - *QuotaEntry*.**QuotaUsed**.

EndIf

If *RemainingQuota* < **Open.File.Volume.FreeSpace**:

OutputBuffer.AvailableAllocationUnits MUST be set to *RemainingQuota* / **Open.File.Volume.ClusterSize**.

EndIf

EndIf

Upon successful completion of the operation, the object store MUST return:

ByteCount MUST be set to ***sizeof(***FILE_FS_SIZE_INFORMATION***)***.

Status set to STATUS_SUCCESS.

3.1.5.12.4 FileFsDeviceInformation

OutputBuffer is of type FILE_FS_DEVICE_INFORMATION, as described in [MS-FSCC] section 2.5.10.

Pseudocode for the operation is as follows:

If **OutputBufferSize** is smaller than ***sizeof(***FILE_FS_DEVICE_INFORMATION***)***, the operation MUST be failed with STATUS_INFO_LENGTH_MISMATCH .

OutputBuffer MUST be filled out as follows:

OutputBuffer.DeviceType set to FILE_DEVICE_DISK or FILE_DEVICE_CD_ROM, as defined in [MS-FSCC] section 2.5.10, depending on the type of media that **Open.File.Volume** is mounted on.

OutputBuffer.Characteristics set to **Open.File.Volume.VolumeCharacteristics**.

Upon successful completion of the operation, the object store MUST return:

ByteCount set to ***sizeof(***FILE_FS_DEVICE_INFORMATION***)***.

Status set to STATUS_SUCCESS.

3.1.5.12.5 FileFsAttributeInformation

OutputBuffer is of type FILE_FS_ATTRIBUTE_INFORMATION, as described in [MS-FSCC] section 2.5.1.

This routine uses the following local variables:

32-bit unsigned integer: *RemainingLength*, *BytesToCopy*

Pseudocode for the operation is as follows:

If **OutputBufferSize** is smaller than ***BlockAlign(FieldOffset(***FILE_FS_ATTRIBUTE_INFORMATION.FileSystemName***)***, 4***)***, the operation MUST be failed with STATUS_INFO_LENGTH_MISMATCH.

OutputBuffer MUST be filled out as follows:

OutputBuffer.FileSystemAttributes set to appropriate values, as specified in [MS-FSCC] section 2.5.1, based on the implementation of the given file system.<100>

OutputBuffer.MaximumComponentNameLength set to different values depending on the file system.<101>

OutputBuffer.FileSystemNameLength set to the length, in bytes, of the name of the file system on **Open.File.Volume**.

Set *RemainingLength* to **OutputBufferSize -** ***FieldOffset(***FILE_FS_ATTRIBUTE_INFORMATION.**FileSystemName*****)***.

If *RemainingLength* < **OutputBuffer.FileSystemNameLength**.

Set *BytesToCopy* to *RemainingLength*.

Else

Set *BytesToCopy* to **OutputBuffer.FileSystemNameLength**.

EndIf

Copy *BytesToCopy* bytes from the file system name string to **OutputBuffer.FileSystemName**.

Upon successful completion of the operation, the object store MUST return:

ByteCount set to ***FieldOffset(***FILE_FS_ATTRIBUTE_INFORMATION.**FileSystemName*****)***+ *BytesToCopy*.

Status set to STATUS_BUFFER_OVERFLOW if *BytesToCopy* < **OutputBuffer.FileSystemNameLength** else STATUS_SUCCESS.

3.1.5.12.6 FileFsControlInformation

OutputBuffer is of type FILE_FS_CONTROL_INFORMATION, as described in [MS-FSCC] section 2.5.2.

Pseudocode for the operation is as follows:

If **OutputBufferSize** is smaller than ***BlockAlign(sizeof(***FILE_FS_CONTROL_INFORMATION***)***, 8***)*** the operation MUST be failed with STATUS_INFO_LENGTH_MISMATCH.

Support for this operation is optional. If the object store does not implement this functionality, the operation MUST be failed with STATUS_INVALID_PARAMETER.

If **Open.File.Volume.IsQuotasSupported** is FALSE, the operation MUST be failed with STATUS_VOLUME_NOT_UPGRADED.

The object store MUST initialize all fields in **OutputBuffer** to zero.

If Quotas are supported on **Open.File.Volume**, the object store MUST set fields in **OutputBuffer** as follows:

OutputBuffer.**DefaultQuotaThreshold** set to **Open**.**File**.**Volume**.**DefaultQuotaThreshold**.

OutputBuffer.**DefaultQuotaLimit** set to **Open**.**File**.**Volume**.**DefaultQuotaLimit**.

OutputBuffer.**FileSystemControlFlags** set to **Open.File.Volume**.**VolumeQuotaState**.

EndIf

Upon successful completion of the operation, the object store MUST return:

ByteCount set to ***sizeof(***FILE_FS_CONTROL_INFORMATION***)***.

Status set to STATUS_SUCCESS.

3.1.5.12.7 FileFsFullSizeInformation

OutputBuffer is of type FILE_FS_FULL_SIZE_INFORMATION, as described in [MS-FSCC] 2.5.4.

This routine uses the following local variables:

64-bit unsigned integer: *RemainingQuota*

FILE_QUOTA_INFORMATION element: *QuotaEntry*

Pseudocode for the operation is as follows:

If **OutputBufferSize** is smaller than ***sizeof(***FILE_FS_FULL_SIZE_INFORMATION***)***, the operation MUST be failed with STATUS_INFO_LENGTH_MISMATCH.

OutputBuffer MUST be filled out as follows:

OutputBuffer.TotalAllocationUnits set to **Open.File.Volume.TotalSpace** / **Open.File.Volume.ClusterSize**.

OutputBuffer.CallerAvailableAllocationUnits set to **Open.File.Volume.FreeSpace** / **Open.File.Volume.ClusterSize**.

OutputBuffer.ActualAvailableAllocationUnits set to **Open.File.Volume.FreeSpace** / **Open.File.Volume.ClusterSize**.

OutputBuffer.SectorsPerAllocationUnit set to **Volume.ClusterSize** / **Open.File.Volume.LogicalBytesPerSector**.

OutputBuffer.BytesPerSector set to **Open.File.Volume. LogicalBytesPerSector**.

If **Open.File.Volume.QuotaInformation** contains an entry *QuotaEntry* that matches the SID of the current **Open**, the object store MUST modify the returned information based on *QuotaEntry* as follows:

If *QuotaEntry*.**QuotaLimit** < **Open.File.Volume.TotalSpace**:

OutputBuffer.TotalAllocationUnits MUST be set to *QuotaEntry*.**QuotaLimit** / **Open.File.Volume.ClusterSize**.

EndIf

If *QuotaEntry*.**QuotaLimit** <= *QuotaEntry*.**QuotaUsed:**

RemainingQuota MUST be set to 0.

Else

RemainingQuota MUST be set to *QuotaEntry*.**QuotaLimit** - *QuotaEntry*.**QuotaUsed**.

EndIf

If *RemainingQuota* < **Open.File.Volume.FreeSpace**:

OutputBuffer.CallerAvailableAllocationUnits MUST be set to *RemainingQuota* / ***Open.File.Volume.ClusterSize***.

EndIf

EndIf

Upon successful completion of the operation, the object store MUST return:

ByteCount set to ***sizeof(****FILE_FS_FULL_SIZE_INFORMATION****)***.

Status set to STATUS_SUCCESS.

3.1.5.12.8 FileFsObjectIdInformation

OutputBuffer is a FILE_FS_OBJECTID_INFORMATION structure as described in [MS-FSCC] section 2.5.6.<102>

Pseudocode for the operation is as follows:

If **OutputBufferSize** is less than ***sizeof(***FILE_FS_OBJECTID_INFORMATION***)***, the operation MUST be failed with STATUS_INFO_LENGTH_MISMATCH.

Support for ObjectIDs is optional. If the object store does not implement this functionality, the operation MUST be failed with STATUS_INVALID_PARAMETER.

If **Open.File.Volume.IsObjectIDsSupported** is FALSE, the operation MUST be failed with STATUS_VOLUME_NOT_UPGRADED.

If **Open.File.Volume.VolumeId** is empty, the operation MUST be failed with STATUS_OBJECT_NAME_NOT_FOUND.

OutputBuffer MUST be filled out as follows:

OutputBuffer.ObjectId set to **Open.File.Volume.VolumeId**.

OutputBuffer.ExtendedInfo set to **Open.File.Volume.ExtendedInfo**.

Upon successful completion of the operation, the object store MUST return:

ByteCount set to ***sizeof(***FILE_FS_OBJECTID_INFORMATION***)***.

Status set to STATUS_SUCCESS.

3.1.5.12.9 FileFsDriverPathInformation

This operation is not supported and MUST be failed with STATUS_NOT_SUPPORTED.

3.1.5.12.10 FileFsSectorSizeInformation

Note: All of the information in this section is subject to change because it applies to a preliminary implementation of the protocol or structure.

OutputBuffer is of type FILE_FS_SECTOR_SIZE_INFORMATION as defined in [MS-FSCC] section 2.5.7.

Pseudocode for the operation is as follows:

If **OutputBufferSize** is smaller than sizeof(FILE_FS_SECTOR_SIZE_INFORMATION), the operation MUST be failed with STATUS_INFO_LENGTH_MISMATCH.

OutputBuffer MUST be filled out as follows:

OutputBuffer.LogicalBytesPerSector set to **Open.Volume.LogicalBytesPerSector**.

OutputBuffer.PhysicalBytesPerSectorForAtomicity is computed as follows:

Set **OutputBuffer.PhysicalBytesPerSectorForAtomicity** to the physical sector size reported from the storage device underlying the object store.

If there was an issue with retrieving the physical sector size information:

Set **OutputBuffer.PhysicalBytesPerSectorForAtomicity** to **Open.Volume.LogicalBytesPerSector**.

ElseIf **OutputBuffer.PhysicalBytesPerSectorForAtomicity** is NOT a power of two, OR

OutputBuffer.PhysicalBytesPerSectorForAtomicity is less than **Open.Volume.LogicalBytesPerSector,** OR

OutputBuffer.PhysicalBytesPerSectorForAtomicity is not a multiple of **Open.Volume.LogicalBytesPerSector**:

Set **OutputBuffer.PhysicalBytesPerSectorForAtomicity** to **Open.Volume.LogicalBytesPerSector**.

EndIf

OutputBuffer.PhysicalBytesPerSectorForPerformance is set to **OutputBuffer.PhysicalBytesPerSectorForAtomicity**.

OutputBuffer.FileSystemEffectivePhysicalBytesPerSectorForAtomicity is computed as follows:

If **OutputBuffer.PhysicalBytesPerSectorForAtomicity** is greater than **Open.Volume.SystemPageSize**:

Set **OutputBuffer.FileSystemEffectivePhysicalBytesPerSectorForAtomicity** to **Open.Volume.SystemPageSize**.

Else:

Set **OutputBuffer.FileSystemEffectivePhysicalBytesPerSectorForAtomicity** to **OutputBuffer.PhysicalBytesPerSectorForAtomicity**.

EndIf

OutputBuffer.BytesOffsetForSectorAlignment is computed as follows:

Set **OutputBuffer.BytesOffsetForSectorAlignment** to the physical offset alignment reported by the storage device.

If there was an issue with retrieving the physical offset alignment:

Set **OutputBuffer.BytesOffsetForSectorAlignment** to SSINFO_OFFSET_UNKNOWN.

EndIf

OutputBuffer.BytesOffsetForPartitionAlignment is computed as follows:

Set **OutputBuffer.BytesOffsetForPartitionAlignment** to **(Open.Volume.PartitionOffset % Open.Volume.LogicalBytesPerSector)**.

OutputBuffer.Flags is set as follows:

Set SSINFO_FLAGS_ALIGNED_DEVICE, SSINFO_FLAGS_PARTITION_ALIGNED_ON_DEVICE flags in **OutputBuffer.Flags**.

If **OutputBuffer.BytesOffsetForSectorAlignment** is not a multiple of **Open.Volume.LogicalBytesPerSector**:

Clear SSINFO_FLAGS_ALIGNED_DEVICE flag in **OutputBuffer.Flags**.

EndIf

If **OutputBuffer.BytesOffsetForPartitionAlignment** is not equal to ((**Open.Volume.LogicalBytesPerSector** – **OutputBuffer.BytesOffsetForPartitionAlignment**) % **Open.Volume.LogicalBytesPerSector**:

Clear SSINFO_FLAGS_PARTITION_ALIGNED_ON_DEVICE flag in **OutputBuffer.Flags**

EndIf

Query the storage device underlying the object store to determine if there is a seek penalty. If there is not a seek penalty, set SSINFO_FLAGS_NO_SEEK_PENALTY flag in **OutputBuffer.Flags**.

Query the storage device underlying the object store to determine if either the TRIM (T13-ATA) or UNMAP (T10-SCSI/SAS) commands are supported. If either command is supported, set SSINFO_FLAGS_TRIM_ENABLED flag in **OutputBuffer.Flags**.

Upon successful completion of the operation, the object store MUST return:

ByteCount set to the size of the FILE_FS_SECTOR_SIZE_INFORMATION structure

Status set to STATUS_SUCCESS.

3.1.5.13 Server Requests a Query of Security Information

The server provides:

Open: The **Open** on which security information is being queried.

OutputBufferSize: The maximum number of bytes to return in **OutputBuffer**.

SecurityInformation: A SECURITY_INFORMATION data type, as defined in [MS-DTYP] section 2.4.7.

On completion, the object store MUST return:

Status: An NTSTATUS code that specifies the result.

OutputBuffer: An array of **OutputBufferSize** bytes formatted as a SECURITY_DESCRIPTOR structure in self-relative format, as described in [MS-DTYP] section 2.4.6.

ByteCount: If the operation returns STATUS_SUCCESS, this will be set to the count of bytes filled into **OutputBuffer**. If the operation returns STATUS_BUFFER_OVERFLOW, this will be set to the required size, in bytes, of **OutputBuffer** so that the security descriptor will fit.

This routine uses the following local variables:

A 32-bit unsigned integer used as a byte index into **OutputBuffer**: *NextFree*

32-bit unsigned integers: *SaclLength*, *MaclLength*

Pseudocode for the operation is as follows:

Let ***sizeof(***SECURITY_DESCRIPTOR_RELATIVE***)*** equal the number of bytes occupied by the **Revision**, **Sbz1**, **Control**, **OffsetOwner**, **OffsetGroup**, **OffsetSacl**, and **OffsetDacl** fields of **OutputBuffer** (that is, the total size of those fields in a SECURITY_DESCRIPTOR in self-relative format, as described in [MS-DTYP] section 2.4.6).

The operation MUST be failed with STATUS_ACCESS_DENIED under either of the following conditions:

SecurityInformation contains any of OWNER_SECURITY_INFORMATION, GROUP_SECURITY_INFORMATION, LABEL_SECURITY_INFORMATION, or DACL_SECURITY_INFORMATION, and **Open.GrantedAccess** does not contain READ_CONTROL.

SecurityInformation contains SACL_SECURITY_INFORMATION and **Open.GrantedAccess** does not contain ACCESS_SYSTEM_SECURITY.

If **Open.Stream.StreamType** is DataStream and **Open.Stream.Name** is not empty, the operation MUST be failed with STATUS_INVALID_PARAMETER; security information may only be queried on a file or directory handle, not on a stream handle.

If **Open.File.SecurityDescriptor** is empty:

If **OutputBufferSize** is smaller than ***sizeof(***SECURITY_DESCRIPTOR_RELATIVE***)***, the object store MUST set **ByteCount** equal to ***sizeof(***SECURITY_DESCRIPTOR_RELATIVE***)***, and the operation MUST be failed with STATUS_BUFFER_OVERFLOW.

The object store MUST set **OutputBuffer.Revision** equal to 1; all other fields of **OutputBuffer** MUST be filled with NULL characters.

The object store MUST set the Self Relative (SR) bit in **OutputBuffer.Control**.

The operation returns STATUS_SUCCESS at this point.

EndIf

Set **ByteCount** equal to ***sizeof(***SECURITY_DESCRIPTOR_RELATIVE***)***.

If **SecurityInformation** contains OWNER_SECURITY_INFORMATION and **Open.File.SecurityDescriptor.Owner** is not NULL:

ByteCount += ***BlockAlign***(**SidLength**(**Open.File.SecurityDescriptor.Owner**), 4)

EndIf

If **SecurityInformation** contains GROUP_SECURITY_INFORMATION and **Open.File.SecurityDescriptor.Group** is not NULL:

ByteCount += ***BlockAlign***(**SidLength** (**Open.File.SecurityDescriptor.Group**), 4)

EndIf

If **SecurityInformation** contains DACL_SECURITY_INFORMATION and the DACL Present (DP) bit is set in **Open.File.SecurityDescriptor.Control** and **Open.File.SecurityDescriptor.Dacl** is not NULL:

ByteCount += ***BlockAlign***(***SidLength***(**Open.File.SecurityDescriptor.Dacl.AclSize**), 4)

EndIf

If **SecurityInformation** contains SACL_SECURITY_INFORMATION|LABEL_SECURITY_INFORMATION and the SACL Present (SP) bit is set in **Open.File.SecurityDescriptor.Control** and

Open.File.SecurityDescriptor.Sacl is not NULL:

SaclLength = ***BlockAlign***(***SidLength***(**Open.File.SecurityDescriptor.Sacl.AclSize**), 4)

ByteCount += *SaclLength*

Else

If **SecurityInformation** contains SACL_SECURITY_INFORMATION and the SACL Present (SP) bit is set in **Open.File.SecurityDescriptor.Control** and **Open.File.SecurityDescriptor.Sacl** is not NULL:

SaclLength = ***BlockAlign***(***SidLength***(**Open.File.SecurityDescriptor.Sacl.AclSize**), 4)

For each access control entry (ACE) (as defined in [MS-DTYP] section 2.4.4) in **Open.File.SecurityDescriptor.Sacl** whose **AceType** field is SYSTEM_MANDATORY_LABEL_ACE_TYPE:

SaclLength -= this ACE's **AceSize** field

EndFor

ByteCount += *SaclLength*

EndIf

If **SecurityInformation** contains LABEL_SECURITY_INFORMATION and the SACL Present (SP) bit is set in **Open.File.SecurityDescriptor.Control** and **Open.File.SecurityDescriptor.Sacl** is not NULL:

MaclLength = ***BlockAlign***((size of ACL as defined in [MS-DTYP] section 2.4.5), 4)

For each ACE (as defined in [MS-DTYP] section 2.4.4) in **Open.File.SecurityDescriptor.Sacl** whose **AceType** field is SYSTEM_MANDATORY_LABEL_ACE_TYPE:

MaclLength += this ACE's **AceSize** field

EndFor

ByteCount += *MaclLength*

EndIf

EndIf

If **ByteCount** is greater than **OutputBufferSize**, the operation MUST be failed with STATUS_BUFFER_OVERFLOW.

The object store MUST set **OutputBuffer.Revision** equal to 1; all other fields of **OutputBuffer** MUST be filled with NULL characters.

The object store MUST set the Self Relative (SR) bit in **OutputBuffer.Control**.

Set *NextFree* to ***sizeof(***SECURITY_DESCRIPTOR_RELATIVE***)*** (that is, to the offset of **OutputBuffer.OwnerSid**).

If **SecurityInformation** contains OWNER_SECURITY_INFORMATION and **Open.File.SecurityDescriptor.Owner** is not NULL:

The object store MUST copy ***SidLength***(**Open.File.SecurityDescriptor.Owner**) bytes from **Open.File.SecurityDescriptor.Owner** to **OutputBuffer** at the position of *NextFree*.

The object store MUST set **OutputBuffer.OffsetOwner** equal to *NextFree*.

The object store MUST set the state of the Owner Defaulted (OD) bit of **OutputBuffer.Control** equal to the state of the same bit in **Open.File.SecurityDescriptor.Control**.

NextFree += ***BlockAlign***(***SidLength***(**Open.File.SecurityDescriptor.Owner**), 4).

EndIf

If **SecurityInformation** contains GROUP_SECURITY_INFORMATION and **Open.File.SecurityDescriptor.Group** is not NULL:

The object store MUST copy ***SidLength***(**Open.File.SecurityDescriptor.Group**) bytes from **Open.File.SecurityDescriptor.Group** to **OutputBuffer** at the position of *NextFree*.

The object store MUST set **OutputBuffer.OffsetGroup** equal to *NextFree*.

The object store MUST set the state of the Group Defaulted (GD) bit of **OutputBuffer.Control** equal to the state of the same bit in **Open.File.SecurityDescriptor.Control**.

NextFree += ***BlockAlign***(***SidLength***(**Open.File.SecurityDescriptor.Group**), 4).

EndIf

If **SecurityInformation** contains DACL_SECURITY_INFORMATION:

The object store MUST set the state of the DACL Present (DP), DACL Defaulted (DD), DACL Protected (PD), and DACL Auto-Inherited (DI) bits of **OutputBuffer.Control** equal to the state of the same bits in **Open.File.SecurityDescriptor.Control**.

If the DACL Present (DP) bit is set in **Open.File.SecurityDescriptor.Control** and **Open.File.SecurityDescriptor.Dacl** is not NULL:

The object store MUST copy **Open.File.SecurityDescriptor.Dacl.AclSize** bytes from **Open.File.SecurityDescriptor.Dacl** to **OutputBuffer** at the position of *NextFree*.

The object store MUST set **OutputBuffer.OffsetDacl** equal to *NextFree*.

NextFree += ***BlockAlign***(**Open.File.SecurityDescriptor.Dacl.AclSize**, 4).

EndIf

EndIf

If **SecurityInformation** contains SACL_SECURITY_INFORMATION|LABEL_SECURITY_INFORMATION:

The object store MUST set the state of the SACL Present (SP), SACL Defaulted (SD), SACL Protected (PS), and SACL Auto-Inherited (SI) bits of **OutputBuffer.Control** equal to the state of the same bits in **Open.File.SecurityDescriptor.Control**.

If the SACL Present (SP) bit is set in **Open.File.SecurityDescriptor.Control** and **Open.File.SecurityDescriptor.Sacl** is not NULL:

The object store MUST copy **Open.File.SecurityDescriptor.Sacl.AclSize** bytes from **Open.File.SecurityDescriptor.Sacl** to **OutputBuffer** at the position of *NextFree*.

The object store MUST set **OutputBuffer.OffsetSacl** equal to *NextFree*.

NextFree += *SaclLength*.

EndIf

Else

If **SecurityInformation** contains SACL_SECURITY_INFORMATION:

The object store MUST set the state of the SACL Present (SP), SACL Defaulted (SD), SACL Protected (PS), and SACL Auto-Inherited (SI) bits of **OutputBuffer.Control** equal to the state of the same bits in **Open.File.SecurityDescriptor.Control**.

If the SACL Present (SP) bit is set in **Open.File.SecurityDescriptor.Control** and **Open.File.SecurityDescriptor.Sacl** is not NULL:

Perform an ACE copy according to the algorithm in section 3.1.5.13.1, setting the ACE copy algorithm's parameters as follows:

DestSacl equal to the position in **OutputBuffer** of *NextFree*.

SrcSacl equal to **Open.File.SecurityDescriptor.Sacl**.

CopyAudit set to TRUE.

The object store MUST set **OutputBuffer.OffsetSacl** equal to *NextFree*.

NextFree += *SaclLength*.

EndIf

Else If **SecurityInformation** contains LABEL_SECURITY_INFORMATION:

The object store MUST set the state of the SACL Present (SP), SACL Defaulted (SD), SACL Protected (PS), and SACL Auto-Inherited (SI) bits of **OutputBuffer.Control** equal to the state of the same bits in **Open.File.SecurityDescriptor.Control**.

If the SACL Present (SP) bit is set in **Open.File.SecurityDescriptor.Control** and **Open.File.SecurityDescriptor.Sacl** is not NULL:

Perform an ACE copy according to the algorithm in section 3.1.5.13.1, setting the ACE copy algorithm's parameters as follows:

DestSacl equal to the position in **OutputBuffer** of *NextFree*.

SrcSacl equal to **Open.File.SecurityDescriptor.Sacl**.

CopyAudit set to FALSE.

The object store MUST set **OutputBuffer.OffsetSacl** equal to *NextFree*.

NextFree += *MaclLength*.

EndIf

EndIf

EndIf

The operation returns STATUS_SUCCESS.

3.1.5.13.1 Algorithm for Copying Audit or Label ACEs Into a Buffer

The inputs for an ACE copy are:

DestSacl: A destination buffer formatted as an access control list (ACL), as defined in [MS-DTYP] section 2.4.5.

SrcSacl: A source buffer formatted as an ACL, as defined in [MS-DTYP] section 2.4.5.

CopyAudit: A Boolean value. If TRUE, this algorithm copies only ACEs whose **AceType** field is not SYSTEM_MANDATORY_LABEL_ACE_TYPE. If FALSE, this algorithm copies only ACEs whose **AceType** field is SYSTEM_MANDATORY_LABEL_ACE_TYPE.

The ACE copy algorithm uses the following local variables:

ACE (as defined in [MS-DTYP] section 2.4.4): *ThisAce*

Byte pointer: *NextFree*

Pseudocode for the algorithm is as follows:

Copy (size of ACL as defined in [MS-DTYP] section 2.4.5) bytes from **SrcSacl** to **DestSacl**.

Set **DestSacl.AceCount** to 0.

Set **DestSacl.AclSize** to (size of ACL as defined in [MS-DTYP] section 2.4.5).

Set *NextFree* to (size of ACL as defined in [MS-DTYP] section 2.4.5) bytes from the beginning of **DestSacl**.

For each ACE *ThisAce* in **SrcSacl**:

If ((**CopyAudit** is TRUE and *ThisAce*.**AceType** is not SYSTEM_MANDATORY_LABEL_ACE_TYPE) or (**CopyAudit** is FALSE and *ThisAce*.**AceType** is SYSTEM_MANDATORY_LABEL_ACE_TYPE)):

Copy *ThisAce*.**AceSize** bytes from *ThisAce* to *NextFree*.

DestSacl.AceCount += 1

DestSacl.AclSize = **DestSacl.AclSize** + *ThisAce*.**AceSize**

Advance *NextFree* by *ThisAce*.**AceSize** bytes.

EndIf

EndFor

3.1.5.14 Server Requests Setting of File Information

The server provides:

Open: An **Open** of a DataFile or DirectoryFile.

FileInformationClass: The type of information being applied, as specified in [MS-FSCC] section 2.4.

InputBuffer: A buffer that contains the information to be applied to the object.

InputBufferSize: The size of the buffer provided.

The object store MUST return:

Status: An NTSTATUS code indicating the result of the operation.

Pseudocode for the operation is as follows:

If **Open.File.Volume.IsReadOnly** is TRUE, the operation MUST be failed with STATUS_MEDIA_WRITE_PROTECTED.

3.1.5.14.1 FileAllocationInformation

Note: Some of the information in this section is subject to change because it applies to a preliminary implementation of the protocol or structure. For information about specific differences between versions, see the behavior notes that are provided in the Product Behavior appendix.

InputBuffer is of type FILE_ALLOCATION_INFORMATION as described in [MS-FSCC] section 2.4.4.

This operation MUST be failed with STATUS_INVALID_PARAMETER under any of the following conditions:

If **Open.Stream.StreamType** is DirectoryStream.

If **InputBuffer.AllocationSize** is greater than the maximum file size allowed by the object store.<103>

Pseudocode for the operation is as follows:

If **Open.GrantedAccess** does not contain FILE_WRITE_DATA, the operation MUST be failed with STATUS_ACCESS_DENIED.

If **Open.Stream.Oplock** is not empty, the object store MUST check for an oplock break according to the algorithm in section 3.1.4.12, with input values as follows:

Open equal to this operation's **Open**

Oplock equal to **Open.Stream.Oplock**

Operation equal to "SET_INFORMATION"

OpParams containing a member **FileInformationClass** containing **FileAllocationInformation**

If the **Oplock** member of the **DirectoryStream** in **Open.Link.ParentFile.StreamList** (hereinafter referred to as *ParentOplock*) is not empty, the object store MUST check for an oplock break on the parent according to the algorithm in section 3.1.4.12, with input values as follows:

Open equal to this operation's **Open**

Oplock equal to *ParentOplock*

Operation equal to "SET_INFORMATION"

OpParams containing a member **FileInformationClass** containing **FileAllocationInformation**

Flags equal to "PARENT_OBJECT"

If **Open.Stream.IsDeleted** is TRUE, the operation SHOULD return STATUS_SUCCESS.

Set *NewAllocationSize* to ***BlockAlign***(**InputBuffer.AllocationSize**,**Open.File.Volume.ClusterSize**) as described in section 3.1.4.5.

If **Open.Stream.AllocationSize** is equal to *NewAllocationSize*, the operation MUST return STATUS_SUCCESS.

If the space for *NewAllocationSize* cannot be reserved in the storage media, then the operation MUST be failed with STATUS_DISK_FULL.

Open.Stream.AllocationSize MUST be set to *NewAllocationSize*.

If *NewAllocationSize* is less than **Open.Stream.Size**:

The object store MUST set **Open.Stream.Size** to *NewAllocationSize*, truncating the Stream.

The object store MUST post a USN change as per section 3.1.4.11 with **File** equal to **File**, **Reason** equal to USN_REASON_DATA_TRUNCATION, and **FileName** equal to **Open.Link.Name**.

EndIf

If **Open.Stream.ValidDataLength** is greater than **Open.Stream.Size**, then the object store MUST set **Open.Stream.ValidDataLength** to **Open.Stream.Size**.

The object store MUST note that the file has been modified as per section 3.1.4.17 with **Open** equal to **Open**.

The operation returns STATUS_SUCCESS.

3.1.5.14.2 FileBasicInformation

Note: Some of the information in this section is subject to change because it applies to a preliminary implementation of the protocol or structure. For information about specific differences between versions, see the behavior notes that are provided in the Product Behavior appendix.

InputBuffer is of type FILE_BASIC_INFORMATION as described in [MS-FSCC] section 2.4.7.

Pseudocode for the operation is as follows:

If **InputBufferSize** is less than **sizeof(**FILE_BASIC_INFORMATION***)***, the operation MUST be failed with STATUS_INFO_LENGTH_MISMATCH.

The operation MUST be failed with STATUS_INVALID_PARAMETER under any of the following conditions:

If **InputBuffer.CreationTime** is less than -1.

If **InputBuffer.LastAccessTime** is less than -1.

If **InputBuffer.LastWriteTime** is less than -1.

If **InputBuffer.ChangeTime** is less than -1.

If **InputBuffer.FileAttributes**.FILE_ATTRIBUTE_DIRECTORY is TRUE and **Open.Stream.StreamType** is DataStream.

If **InputBuffer.FileAttributes**.FILE_ATTRIBUTE_TEMPORARY is TRUE and **Open.File.FileType** is DirectoryFile.

The object store MUST initialize local variables as follows:

CurrentTime to the current system time.

OriginalFileAttributes to **Open.File.FileAttributes**.

UsnReason to 0.

ValidSetAttributes to (FILE_ATTRIBUTE_READONLY | FILE_ATTRIBUTE_HIDDEN | FILE_ATTRIBUTE_SYSTEM | FILE_ATTRIBUTE_ARCHIVE | FILE_ATTRIBUTE_TEMPORARY | FILE_ATTRIBUTE_OFFLINE | FILE_ATTRIBUTE_NOT_CONTENT_INDEXED)

BreakParentOplock to FALSE.

If **InputBuffer.FileAttributes** != 0:

If **Open.File** is equal to **Open.File.Volume.RootDirectory**, the object store MUST NOT allow the application to change the hidden or system attributes:

ValidSetAttributes &= ~(FILE_ATTRIBUTE_HIDDEN | FILE_ATTRIBUTE_SYSTEM)

EndIf

Open.File.FileAttributes &= ~*ValidSetAttributes*

Open.File.FileAttributes |= (**InputBuffer.FileAttributes** & *ValidSetAttributes*)

If **Open.File.FileAttributes** is not equal to *OriginalFileAttributes*:

Set *BreakParentOplock* to TRUE.

The object store MUST set **Open.File.PendingNotifications.**FILE_NOTIFY_CHANGE_ATTRIBUTES to TRUE.

If **InputBuffer.FileAttributes**.FILE_ATTRIBUTE_TEMPORARY is TRUE, the object store MUST set **Open.Stream.IsTemporary** to TRUE; otherwise it MUST be set to FALSE.

If **Open.UserSetChangeTime** is FALSE and **InputBuffer.ChangeTime** != -1, the object store MUST set **Open.File.LastChangeTime** to *CurrentTime*.

If **Open.File.FileAttributes** is not equal to *OriginalFileAttributes*, the object store MUST set *UsnReason*.USN_REASON_BASIC_INFO_CHANGE to TRUE.

If **Open.File.FileAttributes.** FILE_ATTRIBUTE_NOT_CONTENT_INDEXED is not equal to *OriginalFileAttributes*.FILE_ATTRIBUTE_NOT_CONTENT_INDEXED, the object store MUST set *UsnReason*.USN_REASON_INDEXABLE_CHANGE to TRUE.

EndIf

EndIf

If **InputBuffer.ChangeTime** != 0:

The object store MUST set **Open.UserSetChangeTime** to TRUE.

If **InputBuffer.ChangeTime** != -1:

Set *BreakParentOplock* to TRUE.

If **InputBuffer.ChangeTime** !=**Open.File.LastChangeTime**, the object store MUST set *UsnReason*.USN_REASON_BASIC_INFO_CHANGE to TRUE.

The object store MUST set **Open.File.LastChangeTime** to **InputBuffer.ChangeTime**.

EndIf

EndIf

If **InputBuffer.CreationTime** != 0 and **InputBuffer.CreationTime** != -1:

Set *BreakParentOplock* to TRUE.

If **InputBuffer.CreationTime** != **Open.File.CreationTime**, the object store MUST set *UsnReason*.USN_REASON_BASIC_INFO_CHANGE to TRUE.

The object store MUST set **Open.File.CreationTime** to **InputBuffer.CreationTime**.

The object store MUST set **Open.File.PendingNotifications.**FILE_NOTIFY_CHANGE_CREATION to TRUE.

If **Open.UserSetChangeTime** is FALSE and **InputBuffer.ChangeTime** != -1, the object store MUST set **Open.File.LastChangeTime** to *CurrentTime*.

EndIf

If **InputBuffer.LastAccessTime** != 0:

The object store MUST set **Open.UserSetAccessTime** to TRUE.

If **InputBuffer.LastAccessTime** != -1:

Set *BreakParentOplock* to TRUE.

If **InputBuffer. LastAccessTime** != **Open.File.LastAccessTime**, the object store MUST set *UsnReason*.USN_REASON_BASIC_INFO_CHANGE to TRUE.

The object store MUST set **Open.File.LastAccessTime** to **InputBuffer. LastAccessTime**.

The object store MUST set **Open.File.PendingNotifications.**FILE_NOTIFY_CHANGE_LAST_ACCESS to TRUE.

If **Open.UserSetChangeTime** is FALSE and **InputBuffer.ChangeTime** != -1, the object store MUST set **Open.File.LastChangeTime** to *CurrentTime*.

EndIf

EndIf

If **InputBuffer.LastWriteTime** != 0:

The object store MUST set **Open.UserSetModificationTime** to TRUE.

If **InputBuffer.LastWriteTime** != -1:

Set *BreakParentOplock* to TRUE.

If **InputBuffer. LastWriteTime** != **Open.File.LastModificationTime**, the object store MUST set *UsnReason*.USN_REASON_BASIC_INFO_CHANGE to TRUE.

The object store MUST set **Open.File.LastModificationTime** to **InputBuffer. LastWriteTime**.

The object store MUST set **Open.File.PendingNotifications.**FILE_NOTIFY_CHANGE_LAST_WRITE to TRUE.

If **Open.UserSetChangeTime** is FALSE and **InputBuffer.ChangeTime** != -1, the object store MUST set **Open.File.LastChangeTime** to *CurrentTime*.

EndIf

EndIf

If *BreakParentOplock* is TRUE:

If the **Oplock** member of the **DirectoryStream** in **Open.Link.ParentFile.StreamList** (hereinafter referred to as *ParentOplock*) is not empty, the object store MUST check for an oplock break on the parent according to the algorithm in section 3.1.4.12, with input values as follows:

Open equal to this operation's **Open**.

Oplock equal to *ParentOplock*.

Operation equal to "SET_INFORMATION"

OpParams containing a member **FileInformationClass** containing **FileBasicInformation**

Flags equal to "PARENT_OBJECT"

EndIf

The object store MUST post a USN change as per section 3.1.4.11 with **File** equal to **File**, **Reason** equal to *UsnReason*, and **FileName** equal to **Open.Link.Name**.

The operation returns STATUS_SUCCESS.

3.1.5.14.3 FileDispositionInformation

InputBuffer is of type FILE_DISPOSITION_INFORMATION as described in [MS-FSCC] section 2.4.11.

Pseudocode for the operation is as follows:

If **Open.GrantedAccess** does not contain DELETE, the operation MUST be failed with STATUS_ACCESS_DENIED.

If **InputBuffer.DeletePending** is TRUE:

If **File.FileAttributes.FILE_ATTRIBUTE_READONLY** is TRUE, the operation MUST be failed with STATUS_CANNOT_DELETE.

If **Open.Stream.Name** is empty:

If **Open.Stream.StreamType** is DirectoryStream and **Open.File.DirectoryList** is not empty, the operation MUST be failed with STATUS_DIRECTORY_NOT_EMPTY.

Set **Open.Link.IsDeleted** to TRUE.

If **Open.Stream.StreamType** is DirectoryStream:

For each *ChangeNotifyEntry* in **Volume.ChangeNotifyList** where *ChangeNotifyEntry* **.OpenedDirectory.File** is equal to **Open.File** then the following actions MUST be taken:

Remove *ChangeNotifyEntry* from **Volume.ChangeNotifyList**.

Complete the **ChangeNotify** operation with status STATUS_DELETE_PENDING.

EndFor

EndIf

Else:

Set **Open.Stream.IsDeleted** to TRUE.

EndIf

Else:

If **Open.Stream.Name** is empty:

Set Open.Link.IsDeleted to FALSE.

Else:

Set Open.Stream.IsDeleted to FALSE.

EndIf

EndIf

The operation returns STATUS_SUCCESS.

3.1.5.14.4 FileEndOfFileInformation

Note: Some of the information in this section is subject to change because it applies to a preliminary implementation of the protocol or structure. For information about specific differences between versions, see the behavior notes that are provided in the Product Behavior appendix.

InputBuffer is of type FILE_END_OF_FILE_INFORMATION as described in [MS-FSCC] section 2.4.13.

Pseudocode for the operation is as follows:

The operation MUST be failed with STATUS_INVALID_PARAMETER under any of the following conditions:

If **Open.Stream.StreamType** is DirectoryStream.

If **InputBuffer.EndOfFile** is greater than the maximum file size allowed by the object store.<104>

If **Open.GrantedAccess** does not contain FILE_WRITE_DATA, the operation MUST be failed with STATUS_ACCESS_DENIED.

If **Open.Stream.Oplock** is not empty, the object store MUST check for an oplock break according to the algorithm in section 3.1.4.12, with input values as follows:

Open equal to this operation's **Open**

Oplock equal to **Open.Stream.Oplock**

Operation equal to "SET_INFORMATION"

OpParams containing a member **FileInformationClass** containing **FileEndOfFileInformation**

If the **Oplock** member of the **DirectoryStream** in **Open.Link.ParentFile.StreamList** (hereinafter referred to as *ParentOplock*) is not empty, the object store MUST check for an oplock break on the parent according to the algorithm in section 3.1.4.12, with input values as follows:

Open equal to this operation's **Open**

Oplock equal to *ParentOplock*

Operation equal to "SET_INFORMATION"

OpParams containing a member **FileInformationClass** containing **FileEndOfFileInformation**

Flags equal to "PARENT_OBJECT"

If **Open.Stream.IsDeleted** is TRUE, the operation SHOULD return STATUS_SUCCESS.

If **Open.Stream.Size** is equal to **InputBuffer.EndOfFile**, the operation MUST return STATUS_SUCCESS at this point.

If **InputBuffer.EndOfFile** is greater than **Open.Stream.Size:**

The object store MUST post a USN change as per section 3.1.4.11 with **File** equal to **File**, **Reason** equal to USN_REASON_DATA_EXTEND, and **FileName** equal to **Open.Link.Name**.

Else:

The object store MUST post a USN change as per section 3.1.4.11 with **File** equal to **File**, **Reason** equal to USN_REASON_DATA_TRUNCATION, and **FileName** equal to **Open.Link.Name**.

EndIf

If **InputBuffer.EndOfFile** is greater than **Open.Stream.AllocationSize**, the object store MUST set **Open.Stream.AllocationSize** to ***BlockAlign***(**InputBuffer.EndOfFile**, **Open.File.Volume.ClusterSize*)***. If the space cannot be reserved, then the operation MUST be failed with STATUS_DISK_FULL.

If **InputBuffer.EndOfFile** is less than (***BlockAlign***(**Open.Stream.Size**, **Open.File.Volume.ClusterSize**) -**Open.File.Volume.ClusterSize**), the object store SHOULD set **Open.Stream.AllocationSize** to **BlockAlign** (**InputBuffer.EndOfFile**, **Open.File.Volume.ClusterSize**).

If **Open.Stream.ValidDataLength** is greater than **InputBuffer.EndOfFile**, the object store MUST set **Open.Stream.ValidDataLength** to **InputBuffer.EndOfFile**.

The object store MUST set **Open.Stream.Size** to **InputBuffer.EndOfFile**.

The object store MUST note that the file has been modified as per section 3.1.4.17 with **Open** equal to **Open**.

The operation returns STATUS_SUCCESS.

3.1.5.14.5 FileFullEaInformation

InputBuffer is of type FILE_FULL_EA_INFORMATION, as described in [MS-FSCC] section 2.4.15.<105>

Pseudocode for the operation is as follows:

If **Open.File.FileAttributes.FILE_ATTRIBUTE_REPARSE_POINT** is TRUE, the object store MUST fail the operation with STATUS_EAS_NOT_SUPPORTED.

For each *Ea* in **InputBuffer**:

If *Ea*.**EaName** is not well-formed as per [MS-FSCC] 2.4.15, the operation MUST be failed with STATUS_INVALID_EA_NAME.

If *Ea*.**Flags** does not contain a valid set of flags as per [MS-FSCC] 2.4.15, the operation MUST be failed with STATUS_INVALID_EA_NAME.

If *Ea*.**EaName** exists in the **Open.File.ExtendedAttributes,** remove that entry from **Open.File.ExtendedAttributes**, updating **Open.File.ExtendedAttributesLength** to reflect the new list size.

If *Ea*.**EaValueLength** is NOT zero, add *Ea* to **Open.File.ExtendedAttributes**, updating **Open.File.ExtendedAttributesLength** to reflect the new list size

If **Open.File.ExtendedAttributesLength** becomes greater than 64 KB - 5 bytes, the object store MUST fail the operation with STATUS_EA_TOO_LARGE and undo any changes made as part of this operation.

EndFor

If **Open.UserSetChangeTime** is FALSE, the object store MUST update **Open.File.LastChangeTime** to the current time.

The object store MUST set **Open.File.FileAttributes.FILE_ATTRIBUTE_ARCHIVE** to TRUE.

The object store MUST post a USN change as per section 3.1.4.11 with **File** equal to **File**, **Reason** equal to USN_REASON_EA_CHANGE, and **FileName** equal to **Open.Link.Name**.

Set **Open.File.PendingNotifications.**FILE_NOTIFY_CHANGE_EA to TRUE and **Open.File.PendingNotifications.**FILE_NOTIFY_CHANGE_ATTRIBUTES to TRUE.

3.1.5.14.6 FileLinkInformation

Note: Some of the information in this section is subject to change because it applies to a preliminary implementation of the protocol or structure. For information about specific differences between versions, see the behavior notes that are provided in the Product Behavior appendix.

InputBuffer is of type FILE_RENAME_INFORMATION, as described in [MS-FSCC] section 2.4.34.2.<106>

Open represents the pre-existing file to which a new link named in **InputBuffer.FileName** will be created.

Pseudocode for the operation is as follows:

If **Open.Stream.StreamType** is DataStream and **Open.Stream.Name** is not empty, the operation MUST be failed with STATUS_INVALID_PARAMETER.

If **Open.File.FileType** is DirectoryFile, the operation MUST be failed with STATUS_FILE_IS_A_DIRECTORY.

If **Open.Link.IsDeleted** is TRUE, the operation MUST be failed with STATUS_ACCESS_DENIED.

If **InputBuffer.FileName** is not valid as specified in [MS-FSCC] section 2.1.5, the operation MUST be failed with STATUS_OBJECT_NAME_INVALID.

If **Open.File.LinkList** has 1024 or more entries, the operation SHOULD be failed with STATUS_TOO_MANY_LINKS.

Split **InputBuffer.FileName** into *PathName* and *FileName*, as per section 3.1.5.1.

Open *DestinationDirectory* from *PathName*, as per section 3.1.5.1. If the open fails for any reason, the object store MUST fail the request with that error. This request requires that the caller has FILE_ADD_FILE access on the *DestinationDirectory* -- if not, the store MUST fail with STATUS_ACCESS_DENIED.

Search *DestinationDirectory*.**File.DirectoryList** for an *ExistingLink* where *ExistingLink*.**Name** or *ExistingLink*.**ShortName** matches *FileName* using case-sensitivity according to **Open.IsCaseInsensitive**. If such a link is found:

If **InputBuffer.ReplaceIfExists** is TRUE:

Set *ReplacedLinkName* = *DestinationDirectory*.**FileName +** *FileName*.

Remove *ExistingLink* from *ExistingLink*.**File.LinkList**.

Remove *ExistingLink* from *DestinationDirectory*.**File.DirectoryList**.

Set *DeletedLink* to TRUE.

Else:

The operation MUST be failed with STATUS_OBJECT_NAME_COLLISION.

EndIf

EndIf

The object store MUST build a new Link object *NewLink* with fields initialized as follows:

NewLink.**Name** set to *FileName*.

NewLink.**File** set to **Open.File**.

NewLink.**ParentFile** set to *DestinationDirectory*.**File**.

All other fields set to zero.

The object store MUST insert *NewLink* into **Open.File.LinkList**

The object store MUST insert *NewLink* into *DestinationDirectory***.File.DirectoryList**.

The object store MUST update *DestinationDirectory***.File.LastModifiedTime**, *DestinationDirectory***.File.LastAccessedTime**, and *DestinationDirectory***.File.LastChangeTime**.

If the **Oplock** member of the **DirectoryStream** in *DestinationDirectory*.**File.StreamList** (hereinafter referred to as *ParentOplock*) is not empty, the object store MUST check for an oplock break on the parent according to the algorithm in section 3.1.4.12, with input values as follows:

Open equal to this operation's **Open**

Oplock equal to *ParentOplock*

Operation equal to "SET_INFORMATION"

OpParams containing a member **FileInformationClass** containing **FileLinkInformation**

Flags equal to "PARENT_OBJECT"

If **Open.UserSetChangeTime** is FALSE, the object store MUST update **Open.File.LastChangeTime** to the current time.

The object store MUST set **Open.File.FileAttributes**.FILE_ATTRIBUTE_ARCHIVE.

If *DeletedLink* is TRUE:

If *ReplacedLinkName* equals **InputBuffer.FileName** in a case-sensitive comparison:

// In this case, the link name has not changed, but the file it refers to has changed.

Action = FILE_ACTION_MODIFIED

FilterMatch = FILE_NOTIFY_CHANGE_ATTRIBUTES | FILE_NOTIFY_CHANGE_SIZE | FILE_NOTIFY_CHANGE_LAST_WRITE | FILE_NOTIFY_CHANGE_LAST_ACCESS | FILE_NOTIFY_CHANGE_CREATION | FILE_NOTIFY_CHANGE_SECURITY | FILE_NOTIFY_CHANGE_EA

Send directory change notification as per section 3.1.4.1, with **Volume** equal to **File.Volume**, **Action** equal to *Action*, **FilterMatch** equal to *FilterMatch*, and **FileName** equal to **InputBuffer.FileName**.

Else

// In this case, the implementer replaced a link, but the new link created differs only in case.

Action = FILE_ACTION_REMOVED

FilterMatch = FILE_NOTIFY_CHANGE_FILE_NAME

Send directory change notification as per section 3.1.4.1, with **Volume** equal to **File.Volume**, **Action** equal to *Action*, **FilterMatch** equal to *FilterMatch*, and **FileName** equal to **InputBuffer.FileName**.

Action = FILE_ACTION_ADDED

FilterMatch = FILE_NOTIFY_CHANGE_FILE_NAME

Send directory change notification as per section 3.1.4.1, with **Volume** equal to **File.Volume**, **Action** equal to *Action*, **FilterMatch** equal to *FilterMatch*, and **FileName** equal to **InputBuffer.FileName**.

EndIf

Else

// If the implementer did not delete a link, all that needs to be done is to notify that a new link was created.

Action = FILE_ACTION_ADDED

FilterMatch = FILE_NOTIFY_CHANGE_FILE_NAME

Send directory change notification as per section 3.1.4.1, with **Volume** equal to **File.Volume**, **Action** equal to *Action*, **FilterMatch** equal to *FilterMatch*, and **FileName** equal to **InputBuffer.FileName**.

EndIf

The operation returns STATUS_SUCCESS.

3.1.5.14.7 FileModeInformation

InputBuffer is of type FILE_MODE_INFORMATION, as described in [MS-FSCC] section 2.4.24.

Pseudocode for the operation is as follows:

The operation MUST be failed with STATUS_INVALID_PARAMETER under any of the following conditions:

InputBuffer.Mode contains any flag, as defined in [MS-FSCC] section 2.4.24, other than the following:

FILE_WRITE_THROUGH

FILE_SEQUENTIAL_ONLY

FILE_SYNCHRONOUS_IO_ALERT

FILE_SYNCHRONOUS_IO_NONALERT

InputBuffer.Mode contains either FILE_SYNCHRONOUS_IO_ALERT or FILE_SYNCHRONOUS_IO_NONALERT, but **Open.Mode** contains neither FILE_SYNCHRONOUS_IO_ALERT nor FILE_SYNCHRONOUS_IO_NONALERT.

Open.Mode contains either FILE_SYNCHRONOUS_IO_ALERT or FILE_SYNCHRONOUS_IO_NONALERT, but **InputBuffer.Mode** contains neither the FILE_SYNCHRONOUS_IO_ALERT nor FILE_SYNCHRONOUS_IO_NONALERT flags.

InputBuffer.Mode contains both FILE_SYNCHRONOUS_IO_ALERT and FILE_SYNCHRONOUS_IO_NONALERT.

If **Open.Mode** does not contain FILE_NO_INTERMEDIATE_BUFFERING:

If **InputBuffer.Mode** contains FILE_WRITE_THROUGH, set **Open.Mode.FILE_WRITE_THROUGH** to TRUE; otherwise set it to FALSE.

EndIf

If **InputBuffer.Mode** contains FILE_SEQUENTIAL_ONLY, set **Open.Mode.FILE_SEQUENTIAL_ONLY** to TRUE; otherwise set it to FALSE.

If **Open.Mode** contains either FILE_SYNCHRONOUS_IO_ALERT or FILE_SYNCHRONOUS_IO_NONALERT:

If **InputBuffer.Mode** contains FILE_SYNCHRONOUS_IO_ALERT, set **Open.Mode.FILE_SYNCHRONOUS_IO_ALERT** to TRUE; otherwise set it to FALSE.

If **InputBuffer.Mode** contains FILE_SYNCHRONOUS_IO_NONALERT, set **Open.Mode.FILE_SYNCHRONOUS_IO_NONALERT** to TRUE; otherwise set it to FALSE.

EndIf

The operation returns STATUS_SUCCESS.

3.1.5.14.8 FileObjectIdInformation

This operation is not supported and MUST be failed with STATUS_NOT_SUPPORTED.

3.1.5.14.9 FilePositionInformation

InputBuffer is of type FILE_POSITION_INFORMATION, as described in [MS-FSCC] section 2.4.32.

Pseudocode for the operation is as follows:

If **InputBufferSize** is less than the size, in bytes, of the FILE_POSITION_INFORMATION structure, the operation MUST be failed with STATUS_INFO_LENGTH_MISMATCH.

The operation MUST be failed with STATUS_INVALID_PARAMETER under either of the following conditions:

InputBuffer.CurrentByteOffset is less than 0.

Open.Mode contains FILE_NO_INTERMEDIATE_BUFFERING and **InputBuffer.CurrentByteOffset** is not an integer multiple of **Open.File.Volume.LogicalBytesPerSector**.

The object store MUST set Open.**CurrentByteOffset** equal to InputBuffer.CurrentByteOffset.

The operation returns STATUS_SUCCESS.<107>

3.1.5.14.10 FileQuotaInformation

This operation is not supported and MUST be failed with STATUS_NOT_SUPPORTED

3.1.5.14.11 FileRenameInformation

Note: Some of the information in this section is subject to change because it applies to a preliminary implementation of the protocol or structure. For information about specific differences between versions, see the behavior notes that are provided in the Product Behavior appendix.

InputBuffer is of type FILE_RENAME_INFORMATION, as described in [MS-FSCC] section 2.4.34.**Open.FileName** is the pre-existing file name that will be changed by this operation.

This routine uses the following local variables:

Unicode strings: *PathName*, *NewLinkName*, *PrevFullLinkName*, *SourceFullLinkName*

Files: *SourceDirectory*, *DestinationDirectory*

Links: *TargetLink*, *NewLink*

Boolean values (initialized to FALSE): *TargetExistsSameFile*, *ExactCaseMatch*, *MoveToNewDir*, *OverwriteSourceLink*, *RemoveTargetLink*, *FoundLink*, *MatchedShortName*

Boolean values (initialized to TRUE): *ActivelyRemoveSourceLink*, *RemoveSourceLink*, *AddTargetLink*

32-bit unsigned integers: *FilterMatch*, *Action*

Pseudocode for the operation is as follows:

If **Open.GrantedAccess** does not contain DELETE, as defined in [MS-SMB2] section 2.2.13.1, the operation MUST be failed with STATUS_ACCESS_DENIED.

The operation MUST be failed with STATUS_INVALID_PARAMETER under any of the following conditions:

If **InputBuffer.FileNameLength** is equal to zero.

If **InputBuffer.FileNameLength** is an odd number.

If **InputBuffer.FileNameLength** is greater than **InputBufferLength** minus the byte offset into the FILE_RENAME_INFORMATION **InputBuffer** of the **InputBuffer.FileName** field (that is, the total length of **InputBuffer** as given in **InputBufferLength** is insufficient to contain the fixed-size fields of **InputBuffer** plus the length of **InputBuffer.FileName**).

Split **InputBuffer.FileName** into *PathName* and *NewLinkName* per section 3.1.5.1.

If the first character of **InputBuffer.FileName** is '\':

Open *DestinationDirectory* per section 3.1.5.1, setting the open file operation's parameters as follows:

PathName equal to *PathName*.

DesiredAccess equal to FILE_ADD_FILE|SYNCHRONIZE, additionally specifying FILE_ADD_SUBDIRECTORY if **Open.File.FileType** is DirectoryFile.

ShareAccess equal to FILE_SHARE_READ|FILE_SHARE_WRITE.

CreateOptions equal to FILE_OPEN_FOR_BACKUP_INTENT.

CreateDisposition equal to FILE_OPEN.

If open of *DestinationDirectory* fails:

The operation MUST fail with the error returned by the open of *DestinationDirectory*.

Else if *DestinationDirectory*.**Volume** is not equal to **Open.File.Volume**:

The operation MUST be failed with STATUS_NOT_SAME_DEVICE.

EndIf

Else

Set *DestinationDirectory* equal to **Open.Link.ParentFile**.

EndIf

If **Open.Stream.Oplock** is not empty, the object store MUST check for an oplock break according to the algorithm in section 3.1.4.12, with input values as follows:

Open equal to this operation's **Open**.

Oplock equal to **Open.Stream.Oplock**.

Operation equal to "SET_INFORMATION".

OpParams containing a member **FileInformationClass** containing FileRenameInformation.

If the first character of **InputBuffer.FileName** is ':':

Perform a stream rename according to the algorithm in section 3.1.5.14.11.1, setting the stream rename algorithm's parameters as follows:

Pass in the current **Open**.

ReplaceIfExists equal to **InputBuffer.ReplaceIfExists**.

NewStreamName equal to **InputBuffer.FileName**.

If the stream rename algorithm fails, the operation MUST fail with the same status code.

The operation returns STATUS_SUCCESS at this point.

EndIf

If **Open.Link.IsDeleted** is TRUE, the operation MUST be failed with STATUS_ACCESS_DENIED.

If **Open.File.FileType** is DirectoryFile, determine whether **Open.File** contains open files per section 3.1.4.2, with input values as follows:

File equal to **Open.File**.

Open equal to this operation's **Open**.

Operation equal to "SET_INFORMATION".

OpParams containing a member **FileInformationClass** containing FileRenameInformation.

If **Open.File** contains open files, the operation MUST be failed with STATUS_ACCESS_DENIED.

If **InputBuffer.FileName** is not valid as specified in [MS-FSCC] section 2.1.5, the operation MUST be failed with STATUS_OBJECT_NAME_INVALID.

If *DestinationDirectory* is the same as **Open.Link.ParentFile**:

If *NewLinkName* is a case-sensitive exact match with **Open.Link.Name**, the operation MUST return STATUS_SUCCESS at this point.

Else

Set *MoveToNewDir* to TRUE.

EndIf

If *NewLinkName* matches the **Name** or **ShortName** of any **Link** in *DestinationDirectory*.**DirectoryList** using case-sensitivity according to **Open.IsCaseInsensitive**:

Set *FoundLink* to TRUE.

Set *TargetLink* to the existing **Link** found in *DestinationDirectory*.**DirectoryList**. Because the name may have been found using a case-insensitive search (if **Open.IsCaseInsensitive** is TRUE), this preserves the case of the found name.

If *NewLinkName* matched *TargetLink*.**ShortName**, set *MatchedShortName* to TRUE.

Set *RemoveTargetLink* to TRUE.

If *TargetLink*.**File.FileID** equals **Open.File.FileID**, set *TargetExistsSameFile* to TRUE. This detects a rename to another existing link to the same file.

If (*TargetLink*.**Name** is a case-sensitive exact match with *NewLinkName*) or

(*MatchedShortName* is TRUE and

TargetLink.**ShortName** is a case-sensitive exact match with *NewLinkName*):

Set *ExactCaseMatch* to TRUE.

EndIf

If *TargetExistsSameFile* is TRUE:

If *MoveToNewDir* is FALSE:

If **Open.Link.ShortName** is not empty and *TargetLink*.**ShortName** is not empty (this is the case where both the source link and the (existing) requested target are part of the primary link to the same file; this case occurs, for example, in a rename that only changes the case of the name):

Set *ActivelyRemoveSourceLink* to FALSE.

Set *OverwriteSourceLink* to TRUE.

If *ExactCaseMatch* is TRUE, set *RemoveSourceLink* to FALSE (because this algorithm earlier succeeded upon detecting an exact match between the name by which the file was opened and the new requested name, this case only occurs when the file was opened by one half of its primary link, and the requested rename target is the other half; for example, opening a file by its short name and renaming it to its long name).

Else If (**Open.Link.Name** is a case-sensitive exact match with *TargetLink*.**Name**) or

(*MatchedShortName* is TRUE and

Open.Link.Name is a case-sensitive exact match with *TargetLink*.**ShortName**) (this detects the case where the implementer is just changing the case of a single link; for example, given a file with links "primary", "link1", "link2", all in the same directory, the implementer is doing "ren link1 LINK1", and not "ren link1 link2"):

Set *ActivelyRemoveSourceLink* to FALSE.

Set *OverwriteSourceLink* to TRUE.

EndIf

EndIf

If *ExactCaseMatch* is TRUE and

(*OverwriteSourceLink* is FALSE or

Open.IsCaseInsensitive is TRUE or

Open.Link.ShortName is empty)

Set *RemoveTargetLink* and *AddTargetLink* to FALSE.

EndIf

EndIf

If *RemoveTargetLink* is TRUE:

If *TargetExistsSameFile* is FALSE and **InputBuffer.ReplaceIfExists** is FALSE, the operation MUST be failed with STATUS_OBJECT_NAME_COLLISION.

Set *PrevFullLinkName* to the full pathname from **Open.File.Volume.RootDirectory** to *TargetLink*.

If *TargetExistsSameFile* is FALSE:

The operation MUST be failed with STATUS_ACCESS_DENIED under any of the following conditions:

If *TargetLink*.**File.FileType** is DirectoryFile.

If *TargetLink*.**File.FileAttributes.**FILE_ATTRIBUTE_READONLY is TRUE.

If *TargetLink*.**IsDeleted** is TRUE, the operation MUST be failed with STATUS_DELETE_PENDING.

If the caller does not have DELETE access to *TargetLink*.**File**:

If the caller does not have FILE_DELETE_CHILD access to *DestinationDirectory*:

The operation MUST be failed with STATUS_ACCESS_DENIED.

EndIf

EndIf

For each **Stream** on *TargetLink*.**File**:

If *TargetLink*.**File.OpenList** contains an **Open** with a **Stream** matching the current **Stream**, and that **Stream**'s **Oplock** is not empty, the object store MUST check for an oplock break according to the algorithm in section 3.1.4.12, with input values as follows:

Open equal to this operation's **Open**.

Oplock equal to the found **Stream's Oplock**.

Operation equal to SET_INFORMATION.

OpParams containing a member **FileInformationClass** containing **FileEndOfFileInformation**.

If there was not an oplock to be broken and *TargetLink*.**File.OpenList** contains an **Open** with a **Stream** matching the current **Stream**, the operation MUST be failed with STATUS_ACCESS_DENIED.

EndFor

If *TargetLink*.**File.LinkList** contains exactly one element:

The object store MUST delete *TargetLink*.**File** per section 3.1.5.4; if this fails, the operation MUST be failed with the same status.

Else

The object store MUST delete *TargetLink* per section 3.1.5.4; if this fails, the operation MUST be failed with the same status.

The object store MUST post a USN change as per section 3.1.4.11 with **File** equal to **File**, **Reason** equal to (USN_REASON_HARD_LINK_CHANGE | USN_REASON_CLOSE), and **FileName** equal to *TargetLink*.**Name**.

EndIf

Else

The object store MUST post a USN change as per section 3.1.4.11 with **File** equal to **File**, **Reason** equal to USN_REASON_RENAME_OLD_NAME, and **FileName** equal to *TargetLink*.**Name**.

The object store MUST delete *TargetLink* per section 3.1.5.4; if this fails, the operation MUST be failed with the same status.

EndIf

EndIf

EndIf

The object store MUST post a USN change as per section 3.1.4.11 with **File** equal to **File**, **Reason** equal to USN_REASON_RENAME_OLD_NAME, and **FileName** equal to **Open.Link.Name**.

If *RemoveSourceLink* is TRUE:

Set *SourceDirectory* to **Open.Link.ParentFile**.

If *ActivelyRemoveSourceLink* is TRUE:

Remove **Open.Link** from **Open.File.LinkList**.

Remove **Open.Link** from **Open.Link.ParentFile.DirectoryList**.

A new **TunnelCacheEntry** object *TunnelCacheEntry* MUST be constructed and added to the **Open.File.Volume.TunnelCacheList** as follows:

*TunnelCacheEntry***.EntryTime** MUST be set to the current time.

*TunnelCacheEntry***.ParentFile** MUST be set to **Open.Link.ParentFile**.

*TunnelCacheEntry***.FileName** MUST be set to **Open.Link.Name**.

*TunnelCacheEntry***.FileShortName** MUST be set to **Open.Link.ShortName**.

If **Open.FileName** matches **Open.Link.ShortName**, then *TunnelCacheEntry***.KeyByShortName** MUST be set to TRUE, else *TunnelCacheEntry***.KeyByShortName** MUST be set to FALSE.

*TunnelCacheEntry***.FileCreationTime** MUST be set to **Open.File.CreationTime**.

*TunnelCacheEntry***.FileObjectId** MUST be set to **Open.File.ObjectId**.

EndIf

If **Open.File.FileType** is DirectoryFile, then **Open.File** MUST have every **TunnelCacheEntry** associated with it invalidated:

For every *ExistingTunnelCacheEntry* in **Open.File.Volume.TunnelCacheList**:

If *ExistingTunnelCacheEntry***.ParentFile** matches **Open.File**, then *ExistingTunnelCacheEntry* MUST be removed from **Open.File.Volume.TunnelCacheList**.

EndFor

EndIf

EndIf

Set *SourceFullLinkName* to **Open.FileName**.

EndIf

If *AddTargetLink* is TRUE:

The operation MUST be failed with STATUS_ACCESS_DENIED if either of the following conditions are true:

Open.File.FileType is DirectoryFile and the caller does not have FILE_ADD_SUBDIRECTORY access on *DestinationDirectory*.

Open.File.FileType is DataFile and the caller does not have FILE_ADD_FILE access on *DestinationDirectory*.

The object store MUST create a new **Link** object *NewLink*, initialized as follows:

NewLink.**File** equal to **Open.File**.

NewLink.**ParentFile** equal to *DestinationDirectory*.

All other fields set to zero.

If **Open.File.FileType** is DataFile and **Open.IsCaseInsensitive** is TRUE, and tunnel caching is implemented, the object store MUST search **Open.File.Volume.TunnelCacheList** for a *TunnelCacheEntry* where *TunnelCacheEntry***.ParentFile** equals *DestinationDirectory* and either (*TunnelCacheEntry***.KeyByShortName** is FALSE and *TunnelCacheEntry***.FileName** matches *NewLinkName*) or (*TunnelCacheEntry***.KeyByShortName** is TRUE and *TunnelCacheEntry***.FileShortName** matches *NewLinkName*). If such an entry is found:

Set *NewLink***.File.CreationTime** to *TunnelCacheEntry*.**FileCreationTime**.

Set *NewLink***.File.PendingNotifications.** FILE_NOTIFY_CHANGE_CREATION to TRUE.

Set *NewLink***.File.ObjectId** to *TunnelCacheEntry*.**FileObjectId**.

Set *NewLink*.**Name** to *TunnelCacheEntry*.**FileName**.

Set *NewLink*.**ShortName** to *TunnelCacheEntry*.**FileShortName** if that name is not already in use among all names and short names in *NewLink*.**ParentFile.DirectoryList**.

Remove *TunnelCacheEntry* from *NewLink*.**File.Volume.TunnelCacheList**.

Else:

Set *NewLink***.Name** to *NewLinkName*.

EndIf

If **Open.Link.ShortName** is not empty and **Open.IsCaseInsensitive** is TRUE and *NewLink***.ShortName** is empty, then if short names are enabled, the object store MUST create a short name as follows:

If *NewLink***.Name** is 8.3-compliant as described in [MS-FSCC] section 2.1.5.2.1:

Set *NewLink***.ShortName** to *NewLink***.Name**.

Else:

Generate a *NewLink***.ShortName** that is 8.3-compliant as described in [MS-FSCC] section 2.1.5.2.1. The string chosen is implementation-specific, but MUST be unique among all names and short names present in *DestinationDirectory***.DirectoryList**.

EndIf

EndIf

The object store MUST add *NewLink* to *DestinationDirectory*.**DirectoryList**.

The object store MUST replace **Open.Link** with *NewLink*.

If *MoveToNewDir* is TRUE:

*DestinationDirectory***.LastModifiedTime** MUST be updated.

*DestinationDirectory***.LastAccessedTime** MUST be updated.

DestinationDirectory.**LastChangeTime** MUST be updated.

EndIf

EndIf

The object store MUST change the compname component (as specified in [MS-FSCC] section 2.1.5) of **Open.FileName** to *NewLinkName*.

If *RemoveSourceLink* is TRUE:

SourceDirectory.**LastModifiedTime** MUST be updated.

SourceDirectory.**LastAccessedTime** MUST be updated.

SourceDirectory.**LastChangeTime** MUST be updated.

EndIf

The object store MUST update **Open.File.LastChangeTime**.<108>

If **Open.File.FileType** is DataFile, the object store MUST set **Open.File.FileAttributes**.FILE_ATTRIBUTE_ARCHIVE.

FilterMatch = 0

If *RemoveTargetLink* is TRUE and *OverwriteSourceLink* is FALSE and *ExactCaseMatch* is FALSE:

If *TargetLink*.**File.FileType** is DirectoryFile

FilterMatch = FILE_NOTIFY_CHANGE_DIR_NAME

Else

FilterMatch = FILE_NOTIFY_CHANGE_FILE_NAME

EndIf

The object store MUST report a directory change notification per section 3.1.4.1 with **Volume** equal to **Open**.**File.Volume**, **Action** equal to FILE_ACTION_REMOVED, and **FileName** set to *PrevFullLinkName* with a **FilterMatch** of *FilterMatch*.

EndIf

If *RemoveSourceLink* is TRUE:

If **Open.File**.**FileType** is DirectoryFile

FilterMatch = FILE_NOTIFY_CHANGE_DIR_NAME

Else

FilterMatch = FILE_NOTIFY_CHANGE_FILE_NAME

EndIf

If *MoveToNewDir* is TRUE or *AddTargetLink* is FALSE or *RemoveTargetLink* and *ExactCaseMatch* are TRUE: *Action* = FILE_ACTION_REMOVED

Else

Action = FILE_ACTION_REMOVED_OLD_NAME

EndIf

The object store MUST report a directory change notification per section 3.1.4.1 with **Volume** equal to **Open**.**File.Volume**, **Action** equal to *Action*, and **FileName** set to *SourceFullLinkName* with a **FilterMatch** of *FilterMatch*.

EndIf

If *FoundLink* is FALSE or (*OverwriteSourceLink* is TRUE and *ExactCaseMatch* is FALSE) or (*RemoveTargetLink* is TRUE and *ExactCaseMatch* is FALSE):

If *MoveToNewDir* is TRUE, set *Action* to FILE_ACTION_ADDED; otherwise set *Action* to FILE_ACTION_RENAMED_NEW_NAME.

Else If *RemoveTargetLink* is TRUE and *TargetExistsSameFile* is FALSE:

FilterMatch = FILE_NOTIFY_CHANGE_ATTRIBUTES | FILE_NOTIFY_CHANGE_SIZE | FILE_NOTIFY_CHANGE_LAST_WRITE | FILE_NOTIFY_CHANGE_LAST_ACCESS | FILE_NOTIFY_CHANGE_CREATION | FILE_NOTIFY_CHANGE_SECURITY | FILE_NOTIFY_CHANGE_EA

Action = FILE_ACTION_MODIFIED

EndIf

If *FilterMatch* != 0:

The object store MUST report a directory change notification per section 3.1.4.1 with **Volume** equal to **Open**.**File.Volume**, **Action** equal to *Action*, and **FileName** set to **Open.FileName** with a **FilterMatch** of *FilterMatch*.

EndIf

If *MoveToNewDir* is TRUE:

If the **Oplock** member of the **DirectoryStream** in *DestinationDirectory*.**StreamList** (hereinafter referred to as *DestinationParentOplock*) is not empty, the object store MUST check for an oplock break on the parent according to the algorithm in section 3.1.4.12, with input values as follows:

Open equal to this operation's **Open**

Oplock equal to *DestinationParentOplock*

Operation equal to "SET_INFORMATION"

OpParams containing a member **FileInformationClass** containing **FileRenameInformation**

Flags equal to "PARENT_OBJECT"

EndIf

If the **Oplock** member of the **DirectoryStream** in **Open.Link.ParentFile.StreamList** (hereinafter referred to as *SourceParentOplock*) is not empty, the object store MUST check for an oplock break on the parent according to the algorithm in section 3.1.4.12, with input values as follows:

Open equal to this operation's **Open**

Oplock equal to *SourceParentOplock*

Operation equal to "SET_INFORMATION"

OpParams containing a member **FileInformationClass** containing **FileRenameInformation**

Flags equal to "PARENT_OBJECT"

The operation returns STATUS_SUCCESS.

3.1.5.14.11.1 Algorithm for Performing Stream Rename

The inputs for a stream rename are:

Open: an **Open** for the stream being renamed.

ReplaceIfExists: A Boolean value. If TRUE and the target stream exists and the operation is successful, the target stream MUST be replaced. If FALSE and the target stream exists, the operation MUST fail.

NewStreamName: A Unicode string indicating the new name for the stream. This string MUST begin with the Unicode character ":".

The stream rename algorithm uses the following local variables:

Unicode strings: *StreamName*, *StreamTypeName*

Streams: *TargetStream*, *NewDefaultStream*

Pseudocode for the algorithm is as follows:

Split **NewStreamName** into a stream name component *StreamName* and attribute type component *StreamTypeName*, using the character ":" as a delimiter.

The operation MUST be failed with STATUS_INVALID_PARAMETER under any of the following conditions:

The last character of **NewStreamName** is ":".

The character ":" occurs more than three times in **NewStreamName**.

If *StreamName* contains any characters invalid for a streamname as specified in [MS-FSCC] section 2.1.5, or any wildcard characters as defined in section 3.1.4.3.

If *StreamTypeName* contains any characters invalid for a streamname as specified in [MS-FSCC] section 2.1.5, or any wildcard characters as defined in section 3.1.4.3.

Both *StreamName* and *StreamTypeName* are zero-length.

StreamName is more than 255 Unicode characters in length.

If *StreamName* is zero-length and **Open.File.FileType** is DirectoryFile, because a DirectoryFile cannot have an unnamed data stream.

The operation MUST be failed with STATUS_OBJECT_TYPE_MISMATCH if either of the following conditions are true:

Open.Stream.StreamType is DataStream and *StreamTypeName* is not the Unicode string "$DATA".

Open.Stream.StreamType is DirectoryStream and *StreamTypeName* is not the Unicode string "$INDEX_ALLOCATION".

If **Open.Stream.StreamType** is DirectoryStream, the operation MUST be failed with STATUS_INVALID_PARAMETER.

If *StreamName* is a case-insensitive match with **Open.Stream.Name**, the operation MUST return STATUS_SUCCESS at this point.

If the length of *StreamName* is not 0, the object store MUST search **Open.File.StreamList** for a **Stream** with **Stream.Name** matching *StreamName*, ignoring case, setting *TargetStream* to the result.

If *TargetStream* is found:

If **ReplaceIfExists** is FALSE, the operation MUST be failed with STATUS_OBJECT_NAME_COLLISION.

If *TargetStream*.**File.OpenList** contains any Opens to *TargetStream*, the operation MUST be failed with STATUS_INVALID_PARAMETER.

If *TargetStream*.**Size** is not 0, the operation MUST be failed with STATUS_INVALID_PARAMETER.

If *TargetStream*.**AllocationSize** is not 0, the object store SHOULD release any associated allocation and MUST set *TargetStream*.**AllocationSize** to 0.

Else // *TargetStream* is not found:

The object store MUST build a new **Stream** object *TargetStream* with all fields initially set to zero.

Set *TargetStream*.**File** to **Open.File**.

Add *TargetStream* to **Open.File.StreamList**.

EndIf

Set *TargetStream*.**Name** to *StreamName*.

Set *TargetStream*.**Size** to **Open.Stream.Size**.

If **Open.Stream.IsSparse** is TRUE, set *TargetStream*.**IsSparse** to TRUE.

Move **Open.Stream.ExtentList** to *TargetStream*.

Set *TargetStream*.**AllocationSize** to **Open.Stream.AllocationSize**.

If **Open.Stream.Name** is empty, the object store MUST create a new default unnamed stream for the file as follows:

The object store MUST build a new **Stream** object *NewDefaultStream* with all fields initially set to zero.

Set *NewDefaultStream*.**File** to **Open.File**.

Add *NewDefaultStream* to **Open.File.StreamList**.

EndIf

Remove **Open.Stream** from **Open.File.StreamList**.

Set **Open.Stream** to *TargetStream*.

The object store MUST post a USN change as per section 3.1.4.11 with **File** equal to **Open.File**, **Reason** equal to USN_REASON_STREAM_CHANGE, and **FileName** equal to **Open.Link.Name**.

The object store MUST note that the file has been modified as per section 3.1.4.17 with **Open** equal to **Open**.

Return STATUS_SUCCESS.

3.1.5.14.12 FileSfioReserveInformation

This operation is not supported and MUST be failed with STATUS_NOT_SUPPORTED.

3.1.5.14.13 FileShortNameInformation

Note: Some of the information in this section is subject to change because it applies to a preliminary implementation of the protocol or structure. For information about specific differences between versions, see the behavior notes that are provided in the Product Behavior appendix.

InputBuffer is of type FILE_NAME_INFORMATION, as described in [MS-FSCC] section 2.4.37.<109>

Pseudocode for the algorithm is as follows:

If **Open.File.Volume.IsReadOnly** is TRUE, the operation MUST be failed with STATUS_MEDIA_WRITE_PROTECTED.

The operation MUST be failed with STATUS_INVALID_PARAMETER under any of the following conditions:

If **InputBuffer.FileName** starts with '\'.

If **Open.File** is equal to **Open.File.Volume.RootDirectory**.

If **Open.Stream.StreamType** is DataStream and **Open.Stream.Name** is not empty.

If **InputBuffer.FileName** is not a valid 8.3 name as described in [MS-FSCC] section 2.1.5.2.1.

If **Open.IsCaseInsensitive** is FALSE.

The operation MUST be failed with STATUS_ACCESS_DENIED under any of the following conditions:

If **Open.GrantedAccess** contains neither FILE_WRITE_DATA nor FILE_WRITE_ATTRIBUTES as defined in [MS-SMB2] section 2.2.13.1.

If **Open.Link.IsDeleted** is TRUE.

If **Open.Mode.FILE_DELETE_ON_CLOSE** is TRUE.

If **Open.HasRestoreAccess** is FALSE, the operation MUST be failed with STATUS_PRIVILEGE_NOT_HELD.

If **Open.File.Volume.GenerateShortNames** is FALSE, the operation MUST be failed with STATUS_SHORT_NAMES_NOT_ENABLED_ON_VOLUME.

Determine whether **Open.File** contains open files as per section 3.1.4.2, with input values as follows:

File equal to **Open.File**.

Open equal to this operation's **Open**.

Operation equal to "SET_INFORMATION".

OpParams containing a member **FileInformationClass** containing **FileShortNameInformation**.

If **Open.File** contains open files, the operation MUST be failed with STATUS_ACCESS_DENIED.

If **Open.File.FileType** is DirectoryFile:

FilterMatch = FILE_NOTIFY_CHANGE_DIR_NAME

Else

FilterMatch =FILE_NOTIFY_CHANGE_FILE_NAME

EndIf

If **InputBuffer.FileName** is empty:

If **Open.Link.ShortName** is not empty:

OldShortName **= Open.Link.ShortName**.

Set **Open.Link.ShortName** to empty.

Send directory change notification as per section 3.1.4.1, with **Volume** equal to **Open.File.Volume**, **Action** equal to FILE_ACTION_REMOVED, and **FileName** set to *OldShortName* with a **FilterMatch** of *FilterMatch*.

EndIf

Return STATUS_SUCCESS.

EndIf

If **InputBuffer.FileName** equals **Open.Link.ShortName**, return STATUS_SUCCESS.

For each *Link* in **Open.File.LinkList**:

If *Link* is not equal to **Open.Link** and *Link*.**ShortName** is not empty, the operation MUST fail with STATUS_OBJECT_NAME_COLLISION.

EndFor

For each *Link* in **Open.Link.ParentFile.DirectoryList**:

If *Link* is not equal to **Open.Link** and **InputBuffer.FileName** matches *Link*.**Name** or *Link*.**ShortName**, the operation MUST be failed with STATUS_OBJECT_NAME_COLLISION.

EndFor

If **Open.Link.ShortName** is not empty:

Send directory change notification as per section 3.1.4.1, with **Volume** equal to **Open**.**File.Volume**, **Action** equal to FILE_ACTION_RENAMED_OLD_NAME, and **FileName** set to **Open.Link.ShortName** with a **FilterMatch** of *FilterMatch*.

EndIf

If the **Oplock** member of the **DirectoryStream** in **Open.Link.ParentFile.StreamList** (hereinafter referred to as *ParentOplock*) is not empty, the object store MUST check for an oplock break on the parent according to the algorithm in section 3.1.4.12, with input values as follows:

Open equal to this operation's **Open**

Oplock equal to *ParentOplock*

Operation equal to "SET_INFORMATION"

OpParams containing a member **FileInformationClass** containing **FileShortNameInformation**

Flags equal to "PARENT_OBJECT"

Send directory change notification as per section 3.1.4.1, with **Volume** equal to **Open**.**File.Volume**, **Action** equal to FILE_ACTION_RENAMED_NEW_NAME, and **FileName** set to **InputBuffer.FileName** with a **FilterMatch** of *FilterMatch*.

Set **Open.Link.ShortName** to **InputBuffer.FileName**.

The object store MUST update **Open.Link.ParentFile.LastModifiedTime**, **Open.Link.ParentFile.LastAccessedTime**, and **Open.Link.ParentFile.LastChangeTime** to the current time.

If **Open.UserSetChangeTime** is FALSE, the object store MUST update **Open.File.LastChangeTime** to the current time.

If **Open.File.FileType** is DataFile, the object store MUST set **Open.File.FileAttributes**.FILE_ATTRIBUTE_ARCHIVE.

Return STATUS_SUCCESS.

3.1.5.14.14 FileValidDataLengthInformation

Note: Some of the information in this section is subject to change because it applies to a preliminary implementation of the protocol or structure. For information about specific differences between versions, see the behavior notes that are provided in the Product Behavior appendix.

InputBuffer is of type FILE_VALID_DATA_LENGTH_INFORMATION as described in [MS-FSCC] section 2.4.41.<110>

Pseudocode for the operation is as follows:

If **Open.File.Volume.IsReadOnly** is TRUE, the operation MUST be failed with STATUS_MEDIA_WRITE_PROTECTED.

If **Open.HasManageVolumeAccess** is FALSE, the operation MUST be failed with STATUS_PRIVILEGE_NOT_HELD.

The operation MUST be failed with STATUS_INVALID_PARAMETER under any of the following conditions:

If **Open.Stream.ValidDataLength** is greater than **InputBuffer.ValidDataLength**.

If **Open.Stream.IsCompressed** is TRUE.

If **Open.Stream.IsSparse** is TRUE.

If **Open.Stream.Oplock** is not empty, the object store MUST check for an oplock break according to the algorithm in section 3.1.4.12, with input values as follows:

Open equal to this operation's **Open**.

Oplock equal to **Open.Stream.Oplock**.

Operation equal to "SET_INFORMATION".

OpParams containing a member **FileInformationClass** containing **FileValidDataLengthInformation**.

Open.Stream.ValidDataLength MUST be set to **InputBuffer.ValidDataLength**.

Return STATUS_SUCCESS.

3.1.5.15 Server Requests Setting of File System Information

The server provides:

Open: The **Open** on which volume information is being applied.

FsInformationClass: The type of information being applied, as specified in [MS-FSCC] section 2.5.

InputBuffer: A buffer that contains the volume information to be applied to the object.

InputBufferSize: The size of the buffer provided.

The object store MUST return:

Status: An NTSTATUS code indicating the result of the operation.

3.1.5.15.1 FileFsVolumeInformation

This operation is not supported and MUST be failed with STATUS_ INVALID_INFO_CLASS.

3.1.5.15.2 FileFsLabelInformation

This operation is not supported and MUST be failed with STATUS_ INVALID_INFO_CLASS.

3.1.5.15.3 FileFsSizeInformation

This operation is not supported and MUST be failed with STATUS_ INVALID_INFO_CLASS.

3.1.5.15.4 FileFsDeviceInformation

This operation is not supported and MUST be failed with STATUS_INVALID_INFO_CLASS.

3.1.5.15.5 FileFsAttributeInformation

This operation is not supported and MUST be failed with STATUS_ INVALID_INFO_CLASS.

3.1.5.15.6 FileFsControlInformation

InputBuffer is of type FILE_FS_CONTROL_INFORMATION, as described in [MS-FSCC] section 2.5.2.

Pseudocode for the operation is as follows:

If **InputBufferSize** is smaller than ***BlockAlign(sizeof(***FILE_FS_CONTROL_INFORMATION***)***, 8***)*** the operation MUST be failed with STATUS_INVALID_INFO_CLASS.

Support for this operation is optional. If the object store does not implement this functionality, the operation MUST be failed with STATUS_INVALID_PARAMETER.

If **Open.File.Volume.IsQuotasSupported** is FALSE, the operation MUST be failed with STATUS_VOLUME_NOT_UPGRADED.

Open.File.Volume MUST be updated as follows:

Open.File.Volume.DefaultQuotaThreshold set to **InputBuffer.DefaultQuotaThreshold**.

Open.**File**.**Volume**.**DefaultQuotaLimit** set to **InputBuffer.DefaultQuotaLimit**.

Open.**File**.**Volume**.**VolumeQuotaState** set to **InputBuffer.FileSystemControlFlags**. The FILE_VC_QUOTAS_INCOMPLETE and FILE_VC_QUOTAS_REBUILDING flags as well as any undefined flags are cleared from **InputBuffer.FileSystemControlFlags** before being saved.

Upon successful completion of the operation, the object store MUST return:

Status set to STATUS_SUCCESS.

3.1.5.15.7 FileFsFullSizeInformation

This operation is not supported and MUST be failed with STATUS_ INVALID_INFO_CLASS.

3.1.5.15.8 FileFsObjectIdInformation

InputBuffer is a FILE_FS_OBJECTID_INFORMATION structure, as described in [MS-FSCC] section 2.5.6.<111>

Pseudocode for the operation is as follows:

If **InputBufferSize** is less than **sizeof**(FILE_FS_OBJECTID_INFORMATION), the operation MUST be failed with STATUS_INVALID_INFO_CLASS.

Support for ObjectIDs is optional. If the object store does not implement this functionality, the operation MUST be failed with STATUS_INVALID_PARAMETER.

If **Open.File.Volume.IsObjectIDsSupported** is FALSE, the operation MUST be failed with STATUS_VOLUME_NOT_UPGRADED.

Open.File.Volume MUST be updated as follows:

Open.File.Volume.VolumeId set to **InputBuffer.ObjectId**.

Open.File.Volume.ExtendedInfo set to **InputBuffer.ExtendedInfo**.

Upon successful completion of the operation, the object store MUST return:

Status set to STATUS_SUCCESS.

3.1.5.15.9 FileFsDriverPathInformation

This operation is not supported and MUST be failed with STATUS_ INVALID_INFO_CLASS.

3.1.5.15.10 FileFsSectorSizeInformation

This operation is not supported and MUST be failed with STATUS_ INVALID_INFO_CLASS.

3.1.5.16 Server Requests Setting of Security Information

If the object store does not implement security, the operation MUST be failed with STATUS_INVALID_DEVICE_REQUEST.

The server provides:

Open - The **Open** on which security information is being applied.

SecurityInformation - A SECURITY_INFORMATION data type as defined in [MS-DTYP] section 2.4.7.

InputBuffer - A buffer that contains the security descriptor to be applied to the object. The security descriptor is a SECURITY_DESCRIPTOR structure in self-relative format, as described in [MS-DTYP] section 2.4.6.

InputBufferSize - The size of the buffer provided.

On completion, the object store MUST return:

Status - An NTSTATUS code indicating the result of the operation.

This routine uses the following local variables:

Boolean values (initialized to FALSE): *DisableOwnerAces*, *ServerObject*, *DaclUntrusted*

The operation MUST be failed with STATUS_ACCESS_DENIED under any of the following conditions:

SecurityInformation contains any of OWNER_SECURITY_INFORMATION, GROUP_SECURITY_INFORMATION, or LABEL_SECURITY_INFORMATION, and **Open.GrantedAccess** does not contain WRITE_OWNER.

SecurityInformation contains DACL_SECURITY_INFORMATION and **Open.GrantedAccess** does not contain WRITE_DAC.

SecurityInformation contains SACL_SECURITY_INFORMATION and **Open.GrantedAccess** does not contain ACCESS_SYSTEM_SECURITY.

Pseudocode for the operation is as follows:

If **Open.Stream.StreamType** is DataStream and **Open.Stream.Name** is not zero-length, the operation MUST be failed with STATUS_INVALID_PARAMETER; security information may only be set on a file or directory handle, not on a stream handle.

The object store MUST post a USN change as per section 3.1.4.11 with **File** equal to **File**, **Reason** equal to USN_REASON_SECURITY_CHANGE, and **FileName** equal to **Open.Link.Name**.

If the Server Security (SS) bit is set in **InputBuffer.Control**, set *ServerObject* to TRUE, otherwise set it to FALSE.

If the DACL Trusted (DT) bit is set in **InputBuffer.Control**, set *DaclUntrusted* to FALSE, otherwise set it to TRUE.

If **SecurityInformation** contains OWNER_SECURITY_INFORMATION:

If **SecurityInformation** contains DACL_SECURITY_INFORMATION, set *DisableOwnerAces* to FALSE, otherwise set it to TRUE.

If **InputBuffer.OwnerSid** is not present, the operation MUST be failed with STATUS_INVALID_OWNER.

If **InputBuffer.OwnerSid** is not a valid owner SID for a file in the object store, as determined in an implementation-specific manner, the object store MUST return STATUS_INVALID_OWNER.

Else

If **Open.File.SecurityDescriptor.Owner** is NULL, the operation MUST be failed with STATUS_INVALID_OWNER.

EndIf

The object store MUST set **Open.File.SecurityDescriptor** to **InputBuffer**.

If **Open.File.FileType** is not DirectoryFile:

The object store MUST set **Open.File.FileAttributes.**FILE_ATTRIBUTE_ARCHIVE.

The object store MUST update **Open.File.LastChangeTime**.<112>

EndIf

The operation returns STATUS_SUCCESS.

3.1.5.17 Server Requests an Oplock

Note: Some of the information in this section is subject to change because it applies to a preliminary implementation of the protocol or structure. For information about specific differences between versions, see the behavior notes that are provided in the Product Behavior appendix.

The server provides:

Open - The **Open** on which the oplock is being requested.

Type - The type of oplock being requested. Valid values are as follows:

LEVEL_TWO (Corresponds to SMB2_OPLOCK_LEVEL_II as described in [MS-SMB2] section 2.2.13.**)**

LEVEL_ONE (Corresponds to SMB2_OPLOCK_LEVEL_EXCLUSIVE as described in [MS-SMB2] section 2.2.13.)

LEVEL_BATCH (Corresponds to SMB2_OPLOCK_LEVEL_BATCH as described in [MS-SMB2] section 2.2.13.)

LEVEL_GRANULAR (Corresponds to SMB2_OPLOCK_LEVEL_LEASE as described in [MS-SMB2] section 2.2.13.) If this oplock type is specified, the server MUST additionally provide the **RequestedOplockLevel** parameter.

RequestedOplockLevel - A combination of zero or more of the following flags, which are only given for LEVEL_GRANULAR **Type** Oplocks:

READ_CACHING

HANDLE_CACHING

WRITE_CACHING

Following is a list of legal nonzero combinations of **RequestedOplockLevel:**

READ_CACHING

READ_CACHING | WRITE_CACHING

READ_CACHING | HANDLE_CACHING

READ_CACHING | WRITE_CACHING | HANDLE_CACHING

Notes for the operation follow:

If the oplock is not granted, the request completes at this point.

If the oplock is granted, the request does not complete until the oplock is broken; the operation

READ_CACHING

HANDLE_CACHING

WRITE_CACHING

AcknowledgeRequired: A Boolean value; TRUE if the server MUST acknowledge the oplock break, FALSE if not, as specified in section 3.1.5.17.2.

Pseudocode for the operation is as follows:

If **Open.Stream.StreamType** is DirectoryStream:

The operation MUST be failed with STATUS_INVALID_PARAMETER under either of the following conditions:

Type is not LEVEL_GRANULAR.

Type is LEVEL_GRANULAR but **RequestedOplockLevel** is neither READ_CACHING nor (READ_CACHING|HANDLE_CACHING).

If **Type** is LEVEL_EXCLUSIVE or LEVEL_BATCH:

The operation MUST be failed with STATUS_OPLOCK_NOT_GRANTED under either of the following conditions:

Open.File.OpenList contains more than one Open whose **Stream** is the same as **Open.Stream**.

Open.Mode contains either FILE_SYNCHRONOUS_IO_ALERT or FILE_SYNCHRONOUS_IO_NONALERT.

Request an exclusive oplock according to the algorithm in section 3.1.5.17.1, setting the algorithm's parameters as follows:

Pass in the current **Open**.

RequestedOplock equal to **Type**.

The operation MUST at this point return any status code returned by the exclusive oplock request algorithm.

Else If **Type** is LEVEL_TWO:

The operation MUST be failed with STATUS_OPLOCK_NOT_GRANTED under either of the following conditions:

Open.Stream.ByteRangeLockList is not empty.

Open.Mode contains either FILE_SYNCHRONOUS_IO_ALERT or FILE_SYNCHRONOUS_IO_NONALERT.

Request a shared oplock according to the algorithm in section 3.1.5.17.2, setting the algorithm's parameters as follows:

Pass in the current **Open**.

RequestedOplock equal to **Type**.

GrantingInAck equal to FALSE.

The operation MUST at this point return any status code returned by the shared oplock request algorithm.

Else If **Type** is LEVEL_GRANULAR:

If **RequestedOplockLevel** is READ_CACHING or (READ_CACHING|HANDLE_CACHING):

The operation MUST be failed with STATUS_OPLOCK_NOT_GRANTED under either of the following conditions:

Open.Stream.ByteRangeLockList is not empty.

Open.Mode contains either FILE_SYNCHRONOUS_IO_ALERT or FILE_SYNCHRONOUS_IO_NONALERT.

Request a shared oplock according to the algorithm in section 3.1.5.17.2, setting the algorithm's parameters as follows:

Pass in the current **Open**.

RequestedOplock equal to **RequestedOplockLevel**.

GrantingInAck equal to FALSE.

The operation MUST at this point return any status code returned by the shared oplock request algorithm.

Else If **RequestedOplockLevel** is (READ_CACHING|WRITE_CACHING) or (READ_CACHING|WRITE_CACHING|HANDLE_CACHING):

If **Open.Mode** contains either FILE_SYNCHRONOUS_IO_ALERT or FILE_SYNCHRONOUS_IO_NONALERT, the operation MUST be failed with STATUS_OPLOCK_NOT_GRANTED.

Request an exclusive oplock according to the algorithm in section 3.1.5.17.1, setting the algorithm's parameters as follows:

Pass in the current **Open**.

RequestedOplock equal to **RequestedOplockLevel**.

The operation MUST at this point return any status code returned by the exclusive oplock request algorithm.

Else if **RequestedOplockLevel** is 0 (that is, no flags):

The operation MUST return STATUS_SUCCESS at this point.

Else

The operation MUST be failed with STATUS_INVALID_PARAMETER.

EndIf

EndIf

3.1.5.17.1 Algorithm to Request an Exclusive Oplock

Note: Some of the information in this section is subject to change because it applies to a preliminary implementation of the protocol or structure. For information about specific differences between versions, see the behavior notes that are provided in the Product Behavior appendix.

The inputs for requesting an exclusive oplock are:

Open: The **Open** on which the oplock is being requested.

RequestedOplock: The oplock type being requested.

On completion, the object store MUST return:

Status: An NTSTATUS code that specifies the result.

NewOplockLevel: The type of oplock that the requested oplock has been broken to. If a failure status is returned in **Status**, the value of this field is undefined. Valid values are as follows:

LEVEL_NONE (that is, no oplock)

LEVEL_TWO

A combination of one or more of the following flags:

READ_CACHING

HANDLE_CACHING

WRITE_CACHING

AcknowledgeRequired: A Boolean value: TRUE if the server MUST acknowledge the oplock break; FALSE if not, as specified in section 3.1.5.18. If a failure status is returned in **Status**, the value of this field is undefined.

The exclusive oplock request algorithm uses the following local variables:

Boolean value (initialized to FALSE): *GrantExclusiveOplock*

Pseudocode for the algorithm is as follows:

If **Open.Stream.Oplock** is empty:

Build a new **Oplock** object with fields initialized as follows:

Oplock.State set to NO_OPLOCK.

All other fields set to 0/empty.

Store the new **Oplock** object in **Open.Stream.Oplock**.

EndIf

If **Open.Stream.Oplock.State** contains LEVEL_TWO_OPLOCK or NO_OPLOCK:

If **Open.Stream.Oplock.State** contains LEVEL_TWO_OPLOCK and **RequestedOplock** contains one or more of READ_CACHING, HANDLE_CACHING, or WRITE_CACHING, the operation MUST be failed with **Status** set to STATUS_OPLOCK_NOT_GRANTED.

If **Open.Stream.Oplock.State** is equal to LEVEL_TWO_OPLOCK:

Remove the first **Open** *ThisOpen* from **Open.Stream.Oplock.IIOplocks** (there should be exactly one present), and notify the server of an oplock break according to the algorithm in section 3.1.5.17.3, setting the algorithm's parameters as follows:

BreakingOplockOpen equal to *ThisOpen*.

NewOplockLevel equal to LEVEL_NONE.

AcknowledgeRequired equal to FALSE.

OplockCompletionStatus equal to STATUS_SUCCESS.

(The operation does not end at this point; this call to 3.1.5.17.3 completes some earlier call to 3.1.5.17.2.)

EndIf

If **Open.File.OpenList** contains more than one Open whose **Stream** is the same as **Open.Stream**, and NO_OPLOCK is present in **Open.Stream.Oplock.State**, the operation MUST be failed with **Status** set to STATUS_OPLOCK_NOT_GRANTED.

If **Open.Stream.IsDeleted** is TRUE and **RequestedOplock** contains HANDLE_CACHING, the operation MUST be failed with **Status** set to STATUS_OPLOCK_NOT_GRANTED.

Set *GrantExclusiveOplock* to TRUE.

Else If (**Open.Stream.Oplock.State** contains one or more of READ_CACHING, WRITE_CACHING, or HANDLE_CACHING) and

(**Open.Stream.Oplock.State** contains none of BREAK_TO_TWO, BREAK_TO_NONE, BREAK_TO_TWO_TO_NONE, BREAK_TO_READ_CACHING, BREAK_TO_WRITE_CACHING, BREAK_TO_HANDLE_CACHING, or BREAK_TO_NO_CACHING) and (**Open.Stream.Oplock.State.RHBreakQueue** is empty):

// This is a granular oplock and it is not breaking.

If **RequestedOplock** contains none of READ_CACHING, WRITE_CACHING, or HANDLE_CACHING, the operation MUST be failed with **Status** set to STATUS_OPLOCK_NOT_GRANTED.

If **Open.Stream.IsDeleted** is TRUE and **RequestedOplock** contains HANDLE_CACHING, the operation MUST be failed with **Status** set to STATUS_OPLOCK_NOT_GRANTED.

Switch (**Open.Stream.Oplock.State**):

Case READ_CACHING:

If **RequestedOplock** is neither (READ_CACHING|WRITE_CACHING) nor (READ_CACHING|WRITE_CACHING|HANDLE_CACHING), the operation MUST be failed with **Status** set to STATUS_OPLOCK_NOT_GRANTED.

For each **Open** *ThisOpen* in **Open.Stream.Oplock.ROplocks:**

If *ThisOpen*.**TargetOplockKey** != **Open.TargetOplockKey**, the operation MUST be failed with **Status** set to STATUS_OPLOCK_NOT_GRANTED.

EndFor

For each **Open** *ThisOpen* in **Open.Stream.Oplock.ROplocks:**

Remove *ThisOpen* from **Open.Stream.Oplock.ROplocks**.

Notify the server of an oplock break according to the algorithm in section 3.1.5.17.3, setting the algorithm's parameters as follows:

BreakingOplockOpen equal to *ThisOpen*.

NewOplockLevel equal to **RequestedOplock**.

AcknowledgeRequired equal to FALSE.

OplockCompletionStatus equal to STATUS_OPLOCK_SWITCHED_TO_NEW_HANDLE.

(The operation does not end at this point; this call to 3.1.5.17.3 completes some earlier call to 3.1.5.17.2.)

EndFor

Set *GrantExclusiveOplock* to TRUE.

EndCase

Case (READ_CACHING|HANDLE_CACHING):

If **RequestedOplock** is not (READ_CACHING|WRITE_CACHING|HANDLE_CACHING) or **Open.Stream.Oplock.RHBreakQueue** is not empty, the operation MUST be failed with **Status** set to STATUS_OPLOCK_NOT_GRANTED.

For each **Open** *ThisOpen* in **Open.Stream.Oplock.RHOplocks:**

If *ThisOpen*.**TargetOplockKey** != **Open.TargetOplockKey**, the operation MUST be failed with **Status** set to STATUS_OPLOCK_NOT_GRANTED.

EndFor

For each **Open** *ThisOpen* in **Open.Stream.Oplock.RHOplocks:**

Remove *ThisOpen* from **Open.Stream.Oplock.RHOplocks**.

Notify the server of an oplock break according to the algorithm in section 3.1.5.17.3, setting the algorithm's parameters as follows:

BreakingOplockOpen equal to *ThisOpen*.

NewOplockLevel equal to **RequestedOplock**.

AcknowledgeRequired equal to FALSE.

OplockCompletionStatus equal to STATUS_OPLOCK_SWITCHED_TO_NEW_HANDLE.

(The operation does not end at this point; this call to 3.1.5.17.3 completes some earlier call to 3.1.5.17.2.)

EndFor

Set *GrantExclusiveOplock* to TRUE.

EndCase

Case (READ_CACHING|WRITE_CACHING|HANDLE_CACHING|EXCLUSIVE):

If **RequestedOplock** is not (READ_CACHING|WRITE_CACHING|HANDLE_CACHING), the operation MUST be failed with **Status** set to STATUS_OPLOCK_NOT_GRANTED.

// Deliberate FALL-THROUGH to next Case statement.

Case (READ_CACHING|WRITE_CACHING|EXCLUSIVE):

If **RequestedOplock** is neither (READ_CACHING|WRITE_CACHING|HANDLE_CACHING) nor (READ_CACHING|WRITE_CACHING), the operation MUST be failed with **Status** set to STATUS_OPLOCK_NOT_GRANTED.

If **Open.TargetOplockKey** != **Open.Stream.Oplock.ExclusiveOpen.TargetOplockKey**, the operation MUST be failed with **Status** set to STATUS_OPLOCK_NOT_GRANTED.

Notify the server of an oplock break according to the algorithm in section 3.1.5.17.3, setting the algorithm's parameters as follows:

BreakingOplockOpen equal to **Open.Stream.Oplock.ExclusiveOpen**.

NewOplockLevel equal to **RequestedOplock**.

AcknowledgeRequired equal to FALSE.

OplockCompletionStatus equal to STATUS_OPLOCK_SWITCHED_TO_NEW_HANDLE.

(The operation does not end at this point; this call to 3.1.5.17.3 completes some earlier call to 3.1.5.17.1.)

Set **Open.Stream.Oplock.ExclusiveOpen** to NULL.

Set *GrantExclusiveOplock* to TRUE.

EndCase

DefaultCase:

The operation MUST be failed with **Status** set to STATUS_OPLOCK_NOT_GRANTED.

EndSwitch

Else

The operation MUST be failed with **Status** set to STATUS_OPLOCK_NOT_GRANTED.

EndIf

If *GrantExclusiveOplock* is TRUE:

Set **Open.Stream.Oplock.ExclusiveOpen** equal to **Open**.

Set **Open.Stream.Oplock.State** equal to (**RequestedOplock**|EXCLUSIVE).

This operation MUST be made cancelable by inserting it into **CancelableOperations.CancelableOperationList**.

This operation waits until the oplock is broken or canceled, as specified in section 3.1.5.17.3. When the operation specified in section 3.1.5.17.3 is called, its following input parameters are transferred to this routine and then returned by it:

Status is set to **OplockCompletionStatus** from the operation specified in section 3.1.5.17.3.

NewOplockLevel is set to **NewOplockLevel** from the operation specified in section 3.1.5.17.3.

AcknowledgeRequired is set to **AcknowledgeRequired** from the operation specified in section 3.1.5.17.3.

EndIf

3.1.5.17.2 Algorithm to Request a Shared Oplock

Note: Some of the information in this section is subject to change because it applies to a preliminary implementation of the protocol or structure. For information about specific differences between versions, see the behavior notes that are provided in the Product Behavior appendix.

The inputs for requesting a shared oplock are:

Open: The **Open** on which the oplock is being requested.

RequestedOplock: The oplock type being requested.

GrantingInAck: A Boolean value, TRUE if this oplock is being requested as part of an oplock break acknowledgement, FALSE if not.

On completion, the object store MUST return:

Status: An NTSTATUS code that specifies the result.

NewOplockLevel: The type of oplock that the requested oplock has been broken to. If a failure status is returned in **Status**, the value of this field is undefined. Valid values are as follows:

LEVEL_NONE (that is, no oplock)

LEVEL_TWO

A combination of one or more of the following flags:

READ_CACHING

HANDLE_CACHING

WRITE_CACHING

AcknowledgeRequired: A Boolean value: TRUE if the server MUST acknowledge the oplock break; FALSE if not, as specified in section 3.1.5.18. If a failure status is returned in **Status**, the value of this field is undefined.

The shared oplock request algorithm uses the following local variables:

Boolean value (initialized to FALSE): *OplockGranted*

Pseudocode for the algorithm is as follows:

If **Open.Stream.Oplock** is empty:

Build a new **Oplock** object with fields initialized as follows:

Oplock.State set to NO_OPLOCK.

All other fields set to 0/empty.

Store the new **Oplock** object in **Open.Stream.Oplock**.

EndIf

If (**GrantingInAck** is FALSE) and

(**Open.Stream.Oplock.State** contains one or more of BREAK_TO_TWO, BREAK_TO_NONE, BREAK_TO_TWO_TO_NONE, BREAK_TO_READ_CACHING, BREAK_TO_WRITE_CACHING, BREAK_TO_HANDLE_CACHING, BREAK_TO_NO_CACHING, or EXCLUSIVE), then:

The operation MUST be failed with **Status** set to STATUS_OPLOCK_NOT_GRANTED.

EndIf

Switch (**RequestedOplock**):

Case LEVEL_TWO:

The operation MUST be failed with **Status** set to STATUS_OPLOCK_NOT_GRANTED if **Open.Stream.Oplock.State** is anything other than the following:

NO_OPLOCK

LEVEL_TWO_OPLOCK

READ_CACHING

(LEVEL_TWO_OPLOCK|READ_CACHING)

// Deliberate FALL-THROUGH to next Case statement.

Case READ_CACHING:

The operation MUST be failed with **Status** set to STATUS_OPLOCK_NOT_GRANTED if **GrantingInAck** is FALSE and **Open.Stream.Oplock.State** is anything other than the following:

NO_OPLOCK

LEVEL_TWO_OPLOCK

READ_CACHING

(LEVEL_TWO_OPLOCK|READ_CACHING)

(READ_CACHING|HANDLE_CACHING)

(READ_CACHING|HANDLE_CACHING|MIXED_R_AND_RH)

(READ_CACHING|HANDLE_CACHING|BREAK_TO_READ_CACHING)

(READ_CACHING|HANDLE_CACHING|BREAK_TO_NO_CACHING)

If **GrantingInAck** is FALSE:

If there is an **Open** on **Open.Stream.Oplock.RHOplocks** whose **TargetOplockKey** is equal to **Open.TargetOplockKey**, the operation MUST be failed with **Status** set to STATUS_OPLOCK_NOT_GRANTED.

If there is an **Open** on **Open.Stream.Oplock.RHBreakQueue** whose **TargetOplockKey** is equal to **Open.TargetOplockKey**, the operation MUST be failed with **Status** set to STATUS_OPLOCK_NOT_GRANTED.

If there is an **Open** *ThisOpen* on **Open.Stream.Oplock.ROplocks** whose **TargetOplockKey** is equal to **Open.TargetOplockKey** (there should be at most one present):

Remove *ThisOpen* from **Open.Stream.Oplock.ROplocks**.

Notify the server of an oplock break according to the algorithm in section 3.1.5.17.3, setting the algorithm's parameters as follows:

BreakingOplockOpen equal to *ThisOpen*.

NewOplockLevel equal to READ_CACHING.

AcknowledgeRequired equal to FALSE.

OplockCompletionStatus equal to STATUS_OPLOCK_SWITCHED_TO_NEW_HANDLE.

(The operation does not end at this point; this call to 3.1.5.17.3 completes some earlier call to 3.1.5.17.2.)

EndIf

EndIf

If **RequestedOplock** equals LEVEL_TWO:

Add **Open** to **Open.Stream.Oplock.IIOplocks**.

Else // **RequestedOplock** equals READ_CACHING:

Add **Open** to **Open.Stream.Oplock.ROplocks**.

EndIf

Recompute **Open.Stream.Oplock.State** according to the algorithm in section 3.1.4.13, passing **Open.Stream.Oplock** as the **ThisOplock** parameter.

Set *OplockGranted* to TRUE.

EndCase

Case (READ_CACHING|HANDLE_CACHING):

The operation MUST be failed with **Status** set to STATUS_OPLOCK_NOT_GRANTED if **GrantingInAck** is FALSE and **Open.Stream.Oplock.State** is anything other than the following:

NO_OPLOCK

READ_CACHING

(READ_CACHING|HANDLE_CACHING)

(READ_CACHING|HANDLE_CACHING|MIXED_R_AND_RH)

If **Open.Stream.IsDeleted** is TRUE, the operation MUST be failed with **Status** set to STATUS_OPLOCK_NOT_GRANTED.

If **GrantingInAck** is FALSE:

If there is an **Open** *ThisOpen* on **Open.Stream.Oplock.ROplocks** whose **TargetOplockKey** is equal to **Open.TargetOplockKey** (there should be at most one present):

Remove *ThisOpen* from **Open.Stream.Oplocks.ROplocks**.

Notify the server of an oplock break according to the algorithm in section 3.1.5.17.3, setting the algorithm's parameters as follows:

BreakingOplockOpen equal to *ThisOpen*.

NewOplockLevel equal to (READ_CACHING|HANDLE_CACHING).

AcknowledgeRequired equal to FALSE.

OplockCompletionStatus equal to STATUS_OPLOCK_SWITCHED_TO_NEW_HANDLE.

(The operation does not end at this point; this call to 3.1.5.17.3 completes some earlier call to 3.1.5.17.2.)

EndIf

If there is an **Open** *ThisOpen* on **Open.Stream.Oplock.RHOplocks** whose **TargetOplockKey** is equal to **Open.TargetOplockKey** (there should be at most one present):

Notify the server of an oplock break according to the algorithm in section 3.1.5.17.3, setting the algorithm's parameters as follows:

BreakingOplockOpen equal to *ThisOpen*.

NewOplockLevel equal to (READ_CACHING|HANDLE_CACHING).

AcknowledgeRequired equal to FALSE.

OplockCompletionStatus equal to STATUS_OPLOCK_SWITCHED_TO_NEW_HANDLE.

(The operation does not end at this point; this call to 3.1.5.17.3 completes some earlier call to 3.1.5.17.2.)

EndIf

EndIf

Add **Open** to **Open.Stream.Oplock.RHOplocks**.

Recompute **Open.Stream.Oplock.State** according to the algorithm in section 3.1.4.13, passing **Open.Stream.Oplock** as the **ThisOplock** parameter.

Set *OplockGranted* to TRUE.

EndCase

// No other value of **RequestedOplock** is possible.

EndSwitch

If *OplockGranted* is TRUE:

This operation MUST be made cancelable by inserting it into **CancelableOperations.CancelableOperationList**.

The operation waits until the oplock is broken or canceled, as specified in section 3.1.5.17.3. When the operation specified in section 3.1.5.17.3 is called, its following input parameters are transferred to this routine and returned by it:

Status is set to **OplockCompletionStatus** from the operation specified in section 3.1.5.17.3.

NewOplockLevel is set to **NewOplockLevel** from the operation specified in section 3.1.5.17.3.

AcknowledgeRequired is set to **AcknowledgeRequired** from the operation specified in section 3.1.5.17.3.

EndIf

3.1.5.17.3 Indicating an Oplock Break to the Server

The inputs for indicating an oplock break to the server are:

BreakingOplockOpen: The **Open** used to request the oplock that is now breaking.

NewOplockLevel: The type of oplock the requested oplock has been broken to. Valid values are as follows:

LEVEL_NONE (that is, no oplock)

LEVEL_TWO

A combination of one or more of the following flags:

READ_CACHING

HANDLE_CACHING

WRITE_CACHING

AcknowledgeRequired: A Boolean value; TRUE if the server MUST acknowledge the oplock break, FALSE if not, as specified in section 3.1.5.18.

OplockCompletionStatus: The NTSTATUS code to return to the server.

This algorithm simply represents the completion of an oplock request, as specified in section 3.1.5.17.1 or section 3.1.5.17.2. The server is expected to associate the return status from this algorithm with **BreakingOplockOpen**, which is the **Open** passed in when it requested the oplock that is now breaking.

It is important to note that because several oplocks may be outstanding in parallel, although this algorithm represents the completion of an oplock request, it may not result in the completion of the algorithm that called it. In particular, calling this algorithm will result in completion of the caller only if **BreakingOplockOpen** is the same as the **Open** with which the calling algorithm was itself called. To mitigate confusion, each algorithm that refers to this section will specify whether that algorithm's operation terminates at that point or not.

The object store MUST return **OplockCompletionStatus**, **AcknowledgeRequired**, and **NewOplockLevel** to the server (the algorithm is as specified in section 3.1.5.17.1 and section 3.1.5.17.2).

3.1.5.18 Server Acknowledges an Oplock Break

Note: Some of the information in this section is subject to change because it applies to a preliminary implementation of the protocol or structure. For information about specific differences between versions, see the behavior notes that are provided in the Product Behavior appendix.

The server provides:

Open - The **Open** associated with the oplock that has broken.

Type - As part of the acknowledgement, the server indicates a new oplock it would like in place of the one that has broken. Valid values are as follows:

LEVEL_NONE

LEVEL_TWO

LEVEL_GRANULAR - If this oplock type is specified, the server additionally provides:

RequestedOplockLevel - A combination of zero or more of the following flags:

READ_CACHING

HANDLE_CACHING

WRITE_CACHING

If the server requests a new oplock and it is granted, the request does not complete until the oplock is broken; the operation waits for this to happen. Processing of an oplock break is described in section 3.1.5.17.3. Whether the new oplock is granted or not, the object store MUST return:

Status - An NTSTATUS code indicating the result of the operation.

If the server requests a new oplock and it is granted, then when the oplock breaks and the request finally completes, the object store MUST additionally return:

NewOplockLevel: The type of oplock the requested oplock has been broken to. Valid values are as follows:

LEVEL_NONE (that is, no oplock)

LEVEL_TWO

A combination of one or more of the following flags:

READ_CACHING

HANDLE_CACHING

WRITE_CACHING

AcknowledgeRequired: A Boolean value; TRUE if the server MUST acknowledge the oplock break, FALSE if not, as specified in section 3.1.5.17.2.

This routine uses the following local variables:

Boolean values (initialized to FALSE): *NewOplockGranted*, *ReturnBreakToNone*, *FoundMatchingRHOplock*

Pseudocode for the operation is as follows:

If **Open.Stream.Oplock** is empty, the operation MUST be failed with **Status** set to STATUS_INVALID_OPLOCK_PROTOCOL.

If **Type** is LEVEL_NONE or LEVEL_TWO:

If **Open.Stream.Oplock.ExclusiveOpen** is not equal to **Open**, the operation MUST be failed with **Status** set to STATUS_INVALID_OPLOCK_PROTOCOL.

If **Type** is LEVEL_TWO and **Open.Stream.Oplock.State** contains BREAK_TO_TWO:

Set **Open.Stream.Oplock.State** to LEVEL_TWO_OPLOCK.

Set *NewOplockGranted* to TRUE.

Else If **Open.Stream.Oplock.State** contains BREAK_TO_TWO or BREAK_TO_NONE:

Set **Open.Stream.Oplock.State** to NO_OPLOCK.

Else If **Open.Stream.Oplock.State** contains BREAK_TO_TWO_TO_NONE:

Set **Open.Stream.Oplock.State** to NO_OPLOCK.

Set *ReturnBreakToNone* to TRUE.

Else

The operation MUST be failed with **Status** set to STATUS_INVALID_OPLOCK_PROTOCOL.

EndIf

For each **Open** *WaitingOpen* on **Open.Stream.Oplock.WaitList**:

Indicate that the operation associated with *WaitingOpen* may continue according to the algorithm in section 3.1.4.12.1, setting **OpenToRelease** equal to *WaitingOpen*.

Remove *WaitingOpen* from **Open.Stream.Oplock.WaitList**.

EndFor

Set **Open.Stream.Oplock.ExclusiveOpen** to NULL.

If *NewOplockGranted* is TRUE:

The operation waits until the newly-granted Level 2 oplock is broken, as specified in section 3.1.5.17.3.

Else If *ReturnBreakToNone* is TRUE:

In this case the server was expecting the oplock to break to Level 2, but because the oplock is actually breaking to None (that is, no oplock), the object store MUST indicate an oplock break to the server according to the algorithm in section 3.1.5.17.3, setting the algorithm's parameters as follows:

BreakingOplockOpen equal to **Open**.

NewOplockLevel equal to LEVEL_NONE.

AcknowledgeRequired equal to FALSE.

OplockCompletionStatus equal to STATUS_SUCCESS.

(Because **BreakingOplockOpen** is equal to the passed-in **Open**, the operation ends at this point.)

Else

The operation MUST return **Status** set to STATUS_SUCCESS at this point.

EndIf

Else If **Type** is LEVEL_GRANULAR:

Let *BREAK_LEVEL_MASK* = (BREAK_TO_READ_CACHING | BREAK_TO_WRITE_CACHING | BREAK_TO_HANDLE_CACHING | BREAK_TO_NO_CACHING)

Let *R_AND_RH_GRANTED* = (READ_CACHING|HANDLE_CACHING|MIXED_R_AND_RH)

Let *RH_GRANTED* = (READ_CACHING|HANDLE_CACHING)

// If there are no *BREAK_LEVEL_MASK* flags set, this is invalid, unless the

// state is *R_AND_RH_GRANTED* or *RH_GRANTED*, in which case we'll need to see if

// the **RHBreakQueue** is empty.

If (**Open.Stream.Oplock.State** does not contain any flag in *BREAK_LEVEL_MASK* and

(**Open.Stream.Oplock.State** != *R_AND_RH_GRANTED*) and

(**Open.Stream.Oplock.State** != *RH_GRANTED*)) or

(((**Open.Stream.Oplock.State** == *R_AND_RH_GRANTED*) or

(**Open.Stream.Oplock.State** == *RH_GRANTED*)) and

Open.Stream.Oplock.RHBreakQueue is empty):

The request MUST be failed with **Status** set to STATUS_INVALID_OPLOCK_PROTOCOL.

EndIf

Switch **Open.Stream.Oplock.State**

Case (READ_CACHING|HANDLE_CACHING|MIXED_R_AND_RH):

Case (READ_CACHING|HANDLE_CACHING):

Case (READ_CACHING|HANDLE_CACHING|BREAK_TO_READ_CACHING):

Case (READ_CACHING|HANDLE_CACHING|BREAK_TO_NO_CACHING):

For each **RHOpContext** *ThisContext* in **Open.Stream.Oplock.RHBreakQueue**:

If *ThisContext*.**Open** equals **Open**:

Set *FoundMatchingRHOplock* to TRUE.

If *ThisContext*.**BreakingToRead** is FALSE:

If **RequestedOplockLevel** is not 0 and **Open.Stream.Oplock.WaitList** is not empty:

The object store MUST indicate an oplock break to the server according to the algorithm in section 3.1.5.17.3, setting the algorithm's parameters as follows:

BreakingOplockOpen equal to **Open**.

NewOplockLevel equal to LEVEL_NONE.

AcknowledgeRequired equal to TRUE.

OplockCompletionStatus equal to STATUS_CANNOT_GRANT_REQUESTED_OPLOCK.

(Because **BreakingOplockOpen** is equal to the passed-in **Open**, the operation ends at this point.)

EndIf

Else // *ThisContext*.**BreakingToRead** is TRUE.

If **Open.Stream.Oplock.WaitList** is not empty and (**RequestedOplockLevel** is (READ_CACHING|WRITE_CACHING) or (READ_CACHING|WRITE_CACHING|HANDLE_CACHING)):

The object store MUST indicate an oplock break to the server according to the algorithm in section 3.1.5.17.3, setting the algorithm's parameters as follows:

BreakingOplockOpen equal to **Open**.

NewOplockLevel equal to READ_CACHING.

AcknowledgeRequired equal to TRUE.

OplockCompletionStatus equal to STATUS_CANNOT_GRANT_REQUESTED_OPLOCK.

(Because **BreakingOplockOpen** is equal to the passed-in **Open**, the operation ends at this point.)

EndIf

EndIf

Remove *ThisContext* from **Open.Stream.Oplock.RHBreakQueue**.

For each **Open** *WaitingOpen* on **Open.Stream.Oplock.WaitList**:

// The operation waiting for the Read-Handle oplock to break may continue if

// there are no more Read-Handle oplocks outstanding, or if all the remaining

// Read-Handle oplocks have the same oplock key as the waiting operation.

If (**Open.Stream.Oplock.RHBreakQueue** is empty) or (all **RHOpContext.Open.TargetOplockKey** values on **Open.Stream.Oplock.RHBreakQueue** are equal to *WaitingOpen*.**TargetOplockKey**):

Indicate that the operation associated with *WaitingOpen* may continue according to the algorithm in section 3.1.4.12.1, setting **OpenToRelease** equal to *WaitingOpen*.

Remove *WaitingOpen* from **Open.Stream.Oplock.WaitList**.

EndIf

EndFor

If **RequestedOplockLevel** is 0 (that is, no flags):

Recompute **Open.Stream.Oplock.State** according to the algorithm in section 3.1.4.13, passing **Open.Stream.Oplock** as the **ThisOplock** parameter.

The algorithm MUST return **Status** set to STATUS_SUCCESS at this point.

Else If **RequestedOplockLevel** does not contain WRITE_CACHING:

The object store MUST request a shared oplock according to the algorithm in section 3.1.5.17.2, setting the algorithm's parameters as follows:

Pass in the current **Open**.

RequestedOplock equal to **RequestedOplockLevel**.

GrantingInAck equal to TRUE.

The operation MUST at this point return any status code returned by the shared oplock request algorithm.

Else

Set **Open.Stream.Oplock.ExclusiveOpen** to *ThisContext*.**Open**.

Set **Open.Stream.Oplock.State** to (**RequestedOplockLevel**|EXCLUSIVE).

This operation MUST be made cancelable by inserting it into **CancelableOperations.CancelableOperationList**.

This operation waits until the oplock is broken or canceled, as specified in section 3.1.5.17.3.

EndIf

Break out of the For loop.

EndIf

EndFor

If *FoundMatchingRHOplock* is FALSE:

The operation MUST be failed with **Status** set to STATUS_INVALID_OPLOCK_PROTOCOL.

EndIf

The operation returns **Status** set to STATUS_SUCCESS at this point.

EndCase

Case (READ_CACHING|WRITE_CACHING|EXCLUSIVE|BREAK_TO_READ_CACHING):

Case (READ_CACHING|WRITE_CACHING|EXCLUSIVE|BREAK_TO_NO_CACHING):

Case (READ_CACHING|WRITE_CACHING|HANDLE_CACHING|EXCLUSIVE|BREAK_TO_READ_CACHING|BREAK_TO_WRITE_CACHING):

Case (READ_CACHING|WRITE_CACHING|HANDLE_CACHING|EXCLUSIVE|BREAK_TO_READ_CACHING|BREAK_TO_HANDLE_CACHING):

Case (READ_CACHING|WRITE_CACHING|HANDLE_CACHING|EXCLUSIVE|BREAK_TO_READ_CACHING):

Case (READ_CACHING|WRITE_CACHING|HANDLE_CACHING|EXCLUSIVE|BREAK_TO_NO_CACHING):

If **Open.Stream.Oplock.ExclusiveOpen** != **Open**:

The operation MUST be failed with **Status** set to STATUS_INVALID_OPLOCK_PROTOCOL.

EndIf

If **Open.Stream.Oplock.WaitList** is not empty and

Open.Stream.Oplock.State does not contain HANDLE_CACHING and

RequestedOplockLevel is (READ_CACHING|WRITE_CACHING|HANDLE_CACHING):

The object store MUST indicate an oplock break to the server according to the algorithm in section 3.1.5.17.3, setting the algorithm's parameters as follows:

BreakingOplockOpen equal to **Open**.

NewOplockLevel equal to:

(READ_CACHING|WRITE_CACHING) if **Open.Stream.Oplock.State** contains each of BREAK_TO_READ_CACHING and BREAK_TO_WRITE_CACHING and not BREAK_TO_HANDLE_CACHING.

(READ_CACHING|HANDLE_CACHING) if **Open.Stream.Oplock.State** contains each of BREAK_TO_READ_CACHING and BREAK_TO_HANDLE_CACHING and not BREAK_TO_WRITE_CACHING.

READ_CACHING if **Open.Stream.Oplock.State** contains BREAK_TO_READ_CACHING and neither BREAK_TO_WRITE_CACHING nor BREAK_TO_HANDLE_CACHING.

LEVEL_NONE if **Open.Stream.Oplock.State** contains BREAK_TO_NO_CACHING.

AcknowledgeRequired equal to TRUE.

OplockCompletionStatus equal to STATUS_CANNOT_GRANT_REQUESTED_OPLOCK.

(Because **BreakingOplockOpen** is equal to the passed-in **Open**, the operation ends at this point.)

Else

If **Open.Stream.IsDeleted** is TRUE and **RequestedOplockLevel** contains HANDLE_CACHING:

The object store MUST indicate an oplock break to the server according to the algorithm in section 3.1.5.17.3, setting the algorithm's parameters as follows:

BreakingOplockOpen equal to **Open**.

NewOplockLevel equal to **RequestedOplockLevel** without HANDLE_CACHING (for example if **RequestedOplockLevel** is (READ_CACHING|HANDLE_CACHING), then **NewOplockLevel** would be just READ_CACHING).

AcknowledgeRequired equal to TRUE.

OplockCompletionStatus equal to STATUS_CANNOT_GRANT_REQUESTED_OPLOCK.

(Because **BreakingOplockOpen** is equal to the passed-in **Open**, the operation ends at this point.)

EndIf

For each **Open** *WaitingOpen* on **Open.Stream.Oplock.WaitList**:

Indicate that the operation associated with *WaitingOpen* may continue according to the algorithm in section 3.1.4.12.1, setting **OpenToRelease** equal to *WaitingOpen*.

Remove *WaitingOpen* from **Open.Stream.Oplock.WaitList**.

EndFor

If **RequestedOplockLevel** does not contain WRITE_CACHING:

Set **Open.Stream.Oplock.ExclusiveOpen** to NULL.

EndIf

If **RequestedOplockLevel** is 0 (that is, no flags):

Set **Open.Stream.Oplock.State** to NO_OPLOCK.

The operation returns **Status** set to STATUS_SUCCESS at this point.

Else If **RequestedOplockLevel** does not contain WRITE_CACHING:

The object store MUST request a shared oplock according to the algorithm in section 3.1.5.17.2, setting the algorithm's parameters as follows:

Pass in the current **Open**.

RequestedOplock equal to **RequestedOplockLevel**.

GrantingInAck equal to TRUE.

The operation MUST at this point return any status code returned by the shared oplock request algorithm.

Else

// Note that because this oplock is being set up as part of an acknowledgement

// of an exclusive oplock break, **Open.Stream.Oplock.ExclusiveOpen** was set

// at the time of the original oplock request; it contains **Open**.

Set **Open.Stream.Oplock.State** to (**RequestedOplockLevel**|EXCLUSIVE).

This operation MUST be made cancelable by inserting it into **CancelableOperations.CancelableOperationList**.

This operation waits until the oplock is broken or canceled, as specified in section 3.1.5.17.3.

EndIf

EndCase

DefaultCase:

The operation MUST be failed with **Status** set to STATUS_INVALID_OPLOCK_PROTOCOL.

EndSwitch

EndIf

3.1.5.19 Server Requests Canceling an Operation

The server provides:

IORequest: An implementation-specific identifier that is unique for each outstanding IO operation, as described in [MS-CIFS] section 3.3.5.52.

No information is returned.

Cancellation provides the ability for operations that block for extended periods of time to be terminated, thus providing better end-user responsiveness. How operation cancellation is implemented is object store specific.

The Object Store MUST maintain a list of waiting operations that can be canceled by adding them to the **CancelableOperations.CancelableOperationList** as defined in section 3.1.1.12.

Each operation receives an implementation-specific identifier (**IORequest**) that uniquely identifies an in-progress I/O operation, as specified in section 3.1.5.

When a cancellation request is received, scan **CancelableOperations.CancelableOperationList** looking for an operation *CanceledOperation* that matches **IORequest**. If found, *CanceledOperation* MUST be removed from **CancelableOperations.CancelableOperationList** and *CanceledOperation* MUST be failed with STATUS_CANCELED returned for the status of the canceled operation. If not found, the cancel request returns performing no action.<113>

3.1.5.20 Server Requests Querying Quota Information

The server provides:

Open: An Open of a Quota Stream<114>

OutputBufferSize: The maximum number of bytes to return in **OutputBuffer**.

ReturnSingleEntry: A **Boolean** that, if TRUE, indicates at most one entry MUST be returned. If FALSE, one or more entries MAY be returned, up to what will fit in **OutputBufferSize** bytes.

SidList: An optional array of one or more FILE_GET_QUOTA_INFORMATION structures as specified in [MS-FSCC] section 2.4.33.1. This identifies the **SIDs** whose quota information is to be returned.

SidListLength: The length, in bytes, of the **SidList** array. If no **SidList** array is provided, this MUST be set to zero.

StartSid: An optional SID identifying the entry at which to begin scanning quota information. This parameter is ignored if the **SidList** parameter is specified. If no **StartSid** SID is provided, this field is empty.

RestartScan: A **Boolean** that, if TRUE, indicates that enumeration should be restarted from the beginning of the quota list. If FALSE, enumeration should continue from the last position.

On completion, the object store MUST return:

Status: An **NTSTATUS** code that specifies the result.

OutputBuffer: An array of one or more FILE_QUOTA_INFORMATION structures as specified in [MS-FSCC] section 2.4.33.

ByteCount: The number of bytes stored in **OutputBuffer**.

Support for this operation is optional. If the object store does not implement this functionality, the operation MUST be failed with STATUS_INVALID_DEVICE_REQUEST.

Pseudocode for the operation is as follows:

If **SidList** is not empty and **SidListLength** is not a multiple of 4, the operation MUST be failed with STATUS_INVALID_PARAMETER.

If **SidListLength** is not zero but less than *sizeof*(FILE_GET_QUOTA_INFORMATION), **SidList** will be zero filled up to *sizeof*(FILE_GET_QUOTA_INFORMATION).

If **SidList** is not empty:

For each entry in **SidList**, the object store MUST return a FILE_QUOTA_INFORMATION structure as specified in [MS-FSCC] section 2.4.33, where the data returned is from the **Open.Volume.QuotaInformation** entry with the same SID.

If **SidList** includes a SID that does not map to an existing SID in the **Open.Volume.QuotaInformation** list, the object store MUST return a FILE_QUOTA_INFORMATION structure (as specified in [MS-FSCC] section 2.4.33) that is filled with zeros.

If **ReturnSingleEntry** is TRUE, the object store MUST return information only on the first SID in **SidList**. No other **SidList** entries other than the first are processed by the object store.

RestartScan and **StartSid** are ignored.

Else: // **SidList** is empty

If **OutputBufferSize** is less than *sizeof*(FILE_QUOTA_INFORMATION), the operation MUST be failed with STATUS_BUFFER_TOO_SMALL.

If **StartSid** is not empty:

If **StartSid** is not found in **Open.Volume.QuotaInformation** then the operation MUST be failed with STATUS_INVALID_PARAMETER.

Set **Open.LastQuotaId** to the index of the entry in **Open.Volume.QuotaInformation** that matches **StartSid**.

RestartScan is ignored.

Else:

If **RestartScan** is TRUE or **Open.LastQuotaId** is -1:

Set **Open.LastQuotaId** to the index of the first entry in the **Open.Volume.QuotaInformation** list.

Else:

Set **Open.LastQuotaId** to the index of the entry after the current value of **Open.LastQuotaId** of **Open.Volume.QuotaInformation** list.

EndIf

EndIf

The object store MUST return a FILE_QUOTA_INFORMATION structure (as specified in [MS-FSCC] section 2.4.33) that corresponds to the entry in **Open.Volume.QuotaInformationList** that has the index specified by **Open.LastQuotaId**.

If **ReturnSingleEntry** is TRUE, the object store MUST return information on only a single quota entry.

If **ReturnSingleEntry** is FALSE and **Open.LastQuotaId** is not at the end of the **Open.Volume.QuotaInformation** list and more FILE_QUOTA_INFORMATION structures will fit in the remaining **ByteCount**, then more FILE_QUOTA_INFORMATION structures SHOULD be returned until either **Open.LastQuotaId** is at the end of **Open.Volume.QuotaInformation** list or no more FILE_QUOTA_INFORMATION structures will fit in **OutputBuffer**.

The operation MUST fail with STATUS_NO_MORE_ENTRIES when no entries are returned.

Open.LastQuotaId MUST be set to point to the entry in **Open.Volume.QuotaInformation** that represents the last returned FILE_QUOTA_INFORMATION structure in **OutputBuffer**.

EndIf

Upon successful completion, the object store MUST return:

Status set to STATUS_SUCCESS.

ByteCount set to the count, in bytes, of how much data was filled into **OutputBuffer**.

3.1.5.21 Server Requests Setting Quota Information

The server provides:

Open: An **Open** of a Quota Stream<115>.

InputBuffer: A buffer that contains one or more aligned FILE_QUOTA_INFORMATION structures as defined in [MS-FSCC] section 2.4.33.

InputBufferSize: The size, in bytes, of **InputBuffer**.

On completion, the object store MUST return:

Status: An **NTSTATUS** code that specifies the result.

Support for this operation is optional. If the object store does not implement this functionality, the operation MUST be failed with STATUS_INVALID_DEVICE_REQUEST.

Pseudocode for the operation is as follows:

If **InputBufferSize** is zero, the operation MUST be failed with STATUS_INVALID_PARAMETER.

For each FILE_QUOTA_INFORMATION structure *quota* in **InputBuffer:**

Scan **Open.Volume.QuotaInformation** for an entry that matches *quota*.**Sid** and if found, save a pointer in *matchedQuota*; else set *matchedQuota* to empty.

If *quota*.**Sid** == BUILTIN_ADMINISTRATORS (as defined in [MS-DTYP] section 2.4.2.4) and *quota*.**QuotaLimit**

4 Protocol Examples

None.

5 Security

5.1 Security Considerations for Implementers

Security is opaque to file systems. Some file systems store security descriptors as opaque blobs and then call security support routines to perform the necessary security checks. Other file systems do not implement security. Security considerations are called out in the sections where they are used. Please refer to [MS-SECO] for a security overview.

5.2 Index of Security Parameters

Security parameter	Section
SecurityContext	3.1.4.13
SecurityDescriptor	3.1.4.13
SecurityContext	3.1.5.1
SecurityInformation	3.1.5.13
SecurityInformation	3.1.5.16

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:

Microsoft Windows® 2000 operating system

Windows® XP operating system

Windows Server® 2003 operating system

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 1.1: Of the standard Windows file systems, only the UDFS file system supports Software Defect Management.

<2> Section 3.1.1.1: NTFS uses a default cluster size of 4 KB, a maximum cluster size of 64 KB, and a minimum cluster size of 512 bytes. ReFS uses a default cluster size of 64 KB, a maximum cluster size of 128k, and a minimum cluster size of 4 KB. ReFS is supported only on Windows 8 and Windows Server 2012.

<3> Section 3.1.1.1: For AMD64, x86, and ARM systems, this value is 4 KB. For ia64 systems, this value is 8 KB.

<4> Section 3.1.1.1: In NTFS, the CompressionUnitSize is 64 KB for encrypted files, 64 KB for sparse files, and the lesser of 64 KB or (16 * **ClusterSize**) for compressed files. Other file systems do not implement this field.

<5> Section 3.1.1.1: In NTFS, the CompressedChunkSize is 4 KB. Other Windows file systems do not implement this field.

<6> Section 3.1.1.1: Only ReFS supports integrity.

<7> Section 3.1.1.1: Only NTFS supports quotas.

<8> Section 3.1.1.1: This field is present for compatibility with the file level FileObjectIdInformation structure ([MS-FSCC] section 2.4.28). These fields are not currently used by Windows and always contain zeroes.

<9> Section 3.1.1.1: The USN journal is supported on ReFS all versions and NTFS version 3.0 volumes or greater. The USN journal is active by default on Windows client SKUs starting with Windows Vista and later. The USN journal is not active by default on Windows Server SKUs.

<10> Section 3.1.1.1: For Windows 2000, Windows XP, Windows Server 2003, Windows Vista, Windows Server 2008, Windows 7, and Windows Server 2008 R2, the maximum file size of a file on an NTFS volume is the smaller of $(2^{32} - 1)$ * cluster size, and 16 terabytes (TB). For Windows 8 and Windows Server 2012, the maximum file size of file on a NTFS volume is $(2^{32} - 1)$ * cluster size. For example, if the cluster size is 512 bytes, the maximum file size is 2 TB.

<11> Section 3.1.1.2: ReFS does not implement the TunnelCache.

<12> Section 3.1.1.3: ReFS and exFAT do not implement **ShortNames**.

<13> Section 3.1.1.3: The following table defines the support of file time stamps across various Windows file systems. More information can be found in section 6 of the File System Behavior Overview document [FSBO].

Timestamp	ReFS	NTFS	FAT	EXFAT	UDFS
CreationTime	Stored in UTC 100 nanosecond granularity	Stored in UTC 100 nanosecond granularity	Stored in local time 10 millisecond granularity	Stored in UTC if available, else in local time 10 millisecond granularity	Stored in UTC if available, else in local time 1 microsecond granularity
LastAccessTime	Stored in UTC 100 nanosecond granularity Updated at 60 minute granularity	Stored in UTC 100 nanosecond granularity Updated at 60 minute granularity	Stored in local time 1 day granularity	Stored in UTC if available, else in local time 2 second granularity	Stored in UTC if available, else in local time 1 microsecond granularity
ChangeTime	Stored in UTC 100 nanosecond granularity	Stored in UTC 100 nanosecond granularity	Not Supported	Not Supported	Stored in UTC if available, else in local time 1 microsecond granularity
LastWriteTime	Stored in UTC 100 nanosecond granularity	Stored in UTC 100 nanosecond granularity	Stored in local time 2 second granularity	Stored in UTC if available, else in local time 10 millisecond granularity	Stored in UTC if available, else in local time 1 microsecond granularity

<14> Section 3.1.1.3: The following table defines the support of file time stamps across various Windows file systems. More information can be found in section 6 of the File System Behavior Overview document [FSBO].

Timestamp	ReFS	NTFS	FAT	EXFAT	UDFS
CreationTime	Stored in UTC 100 nanosecond granularity	Stored in UTC 100 nanosecond granularity	Stored in local time 10 millisecond granularity	Stored in UTC if available, else in local time 10 millisecond granularity	Stored in UTC if available, else in local time 1 microsecond granularity
LastAccessTime	Stored in UTC 100 nanosecond granularity Updated at 60 minute granularity	Stored in UTC 100 nanosecond granularity Updated at 60 minute granularity	Stored in local time 1 day granularity	Stored in UTC if available, else in local time 2 second granularity	Stored in UTC if available, else in local time 1 microsecond granularity
ChangeTime	Stored in UTC 100 nanosecond granularity	Stored in UTC 100 nanosecond granularity	Not Supported	Not Supported	Stored in UTC if available, else in local time 1 microsecond granularity
LastWriteTime	Stored in UTC 100 nanosecond granularity	Stored in UTC 100 nanosecond granularity	Stored in local time 2 second granularity	Stored in UTC if available, else in local time 10 millisecond granularity	Stored in UTC if available, else in local time 1 microsecond granularity

<15> Section 3.1.1.3: The following table defines the support of file time stamps across various Windows file systems. More information can be found in section 6 of the File System Behavior Overview document [FSBO].

Timestamp	ReFS	NTFS	FAT	EXFAT	UDFS
CreationTime	Stored in UTC 100 nanosecond granularity	Stored in UTC 100 nanosecond granularity	Stored in local time 10 millisecond granularity	Stored in UTC if available, else in local time 10 millisecond granularity	Stored in UTC if available, else in local time 1 microsecond granularity
LastAccessTime	Stored in UTC 100 nanosecond granularity Updated at 60 minute granularity	Stored in UTC 100 nanosecond granularity Updated at 60 minute granularity	Stored in local time 1 day granularity	Stored in UTC if available, else in local time 2 second granularity	Stored in UTC if available, else in local time 1 microsecond granularity
ChangeTime	Stored in UTC 100 nanosecond granularity	Stored in UTC 100 nanosecond granularity	Not Supported	Not Supported	Stored in UTC if available, else in local time 1 microsecond

Timestamp	ReFS	NTFS	FAT	EXFAT	UDFS
					granularity
LastWriteTime	Stored in UTC 100 nanosecond granularity	Stored in UTC 100 nanosecond granularity	Stored in local time 2 second granularity	Stored in UTC if available, else in local time 10 millisecond granularity	Stored in UTC if available, else in local time 1 microsecond granularity

<16> Section 3.1.1.3: In Windows Vista/Windows Server 2008 and later, LastAccessTime updates are disabled by default in the ReFS and NTFS file systems. It is only updated when the file is closed. This behavior is controlled by the following registry key: HKLM\System\CurrentControlSet\Control\FileSystem\NtfsDisableLastAccessUpdate. A nonzero value means LastAccessTime updates are disabled. A value of zero means they are enabled.

<17> Section 3.1.1.3: The following table defines the support of file time stamps across various Windows file systems. More information can be found in section 6 of the File System Behavior Overview document [FSBO].

Timestamp	ReFS	NTFS	FAT	EXFAT	UDFS
CreationTime	Stored in UTC 100 nanosecond granularity	Stored in UTC 100 nanosecond granularity	Stored in local time 10 millisecond granularity	Stored in UTC if available, else in local time 10 millisecond granularity	Stored in UTC if available, else in local time 1 microsecond granularity
LastAccessTime	Stored in UTC 100 nanosecond granularity Updated at 60 minute granularity	Stored in UTC 100 nanosecond granularity Updated at 60 minute granularity	Stored in local time 1 day granularity	Stored in UTC if available, else in local time 2 second granularity	Stored in UTC if available, else in local time 1 microsecond granularity
ChangeTime	Stored in UTC 100 nanosecond granularity	Stored in UTC 100 nanosecond granularity	Not Supported	Not Supported	Stored in UTC if available, else in local time 1 microsecond granularity
LastWriteTime	Stored in UTC 100 nanosecond granularity	Stored in UTC 100 nanosecond granularity	Stored in local time 2 second granularity	Stored in UTC if available, else in local time 10 millisecond granularity	Stored in UTC if available, else in local time 1 microsecond granularity

<18> Section 3.1.1.3: Only NTFS implements EAs.

<19> Section 3.1.1.3: Only NTFS implements EAs.

<20> Section 3.1.1.3: Only NTFS implements object IDs.

<21> Section 3.1.1.3: Only NTFS implements object IDs.

<22> Section 3.1.1.3: Only NTFS and UDFS implement named streams.

<23> Section 3.1.1.3: ReFS and exFAT do not implement **ShortNames**.

<24> Section 3.1.1.3: Only NTFS implements encryption.

<25> Section 3.1.1.4: For ReFS, there will always be exactly one link per file or directory.

<26> Section 3.1.1.4: On ReFS or exFAT, this field MUST be empty.

<27> Section 3.1.1.5: Only NTFS supports compression.

<28> Section 3.1.1.5: Only ReFS supports integrity.

<29> Section 3.1.1.5: Only ReFS supports integrity.

<30> Section 3.1.1.5: Only NTFS and UDFS support sparse files.

<31> Section 3.1.1.5: Only NTFS supports encryption.

<32> Section 3.1.1.6: Only NTFS implements EAs.

<33> Section 3.1.5.1.1: For the NTFS file system the **FileID** consists of a 48-bit index into the MFT (the low 48 bit bits) and a 16-bit sequence number (the high 16 bits).

<34> Section 3.1.5.1.1: For the NTFS file system this is the index portion (low 48 bits) of the **FileID**.

<35> Section 3.1.5.1.1: Only ReFS supports FILE_ATTRIBUTE_INTEGRITY_STREAM.

<36> Section 3.1.5.1.1: Only NTFS and ReFS support FILE_ATTRIBUTE_NO_SCRUB_DATA.

<37> Section 3.1.5.1.1: Only NTFS and UDFS implement named streams.

<38> Section 3.1.5.5.1: This directory is only available on NTFS volumes formatted to NTFS version 3.0 or late.

<39> Section 3.1.5.5.1: "*" is treated as 0x0000002A during the search, and it gives the practical behavior of a wildcard since an ObjectId starts with a much larger value. Similarly, "?" is treated as 0x0000003F and so practically it behaves like "*".

<40> Section 3.1.5.5.2: This directory is only available on NTFS volumes formatted to NTFS version 3.0 or later.

<41> Section 3.1.5.5.3.1: For ReFS, this value MUST be zero.

<42> Section 3.1.5.5.3.3: For ReFS, this value MUST be zero.

<43> Section 3.1.5.5.3.4: For ReFS, this value MUST be zero.

<44> Section 3.1.5.5.3.5: For ReFS, this value MUST be zero.

<45> Section 3.1.5.6: This is only implemented by the NTFS file system. Other file systems return STATUS_SUCCESS and perform no other action.

<46> Section 3.1.5.9.1: This is only implemented by the NTFS file system.

<47> Section 3.1.5.9.1: If the generated ObjectId collides with existing ObjectIds on the volume, Windows retries up to 16 times before failing the operation with STATUS_DUPLICATE_NAME.

<48> Section 3.1.5.9.1: The file system only updates LastChangeTime if no user has explicitly set LastChangeTime. Some Windows file systems defer setting the LastChangeTime until the handle is closed.

<49> Section 3.1.5.9.2: This is only implemented by the NTFS file system.

<50> Section 3.1.5.9.2: The file system only updates LastChangeTime if no user has explicitly set LastChangeTime. Some Windows file systems defer setting the LastChangeTime until the handle is closed.

<51> Section 3.1.5.9.3: This is only implemented by the NTFS file system.

<52> Section 3.1.5.9.3: The file system only updates LastChangeTime if no user has explicitly set LastChangeTime. Some Windows file systems defer setting the LastChangeTime until the handle is closed.

<53> Section 3.1.5.9.5: This is only implemented by the ReFS, NTFS, FAT, and exFAT file systems.

<54> Section 3.1.5.9.5: The NTFS file system sets an NTFS_STATISTICS structure as specified in [MS-FSCC] section 2.3.8.2. The FAT file system sets a FAT_STATISTICS structure as specified in [MS-FSCC] section 2.3.8.3. The EXFAT file system sets a EXFAT_STATISTICS structure as specified in [MS-FSCC] section 2.3.8.4.

<55> Section 3.1.5.9.6: This is only implemented by the NTFS file system.

<56> Section 3.1.5.9.6: Some file systems have more efficient mechanisms to obtain a list of files. For instance, NTFS iterates through all base file records of the MFT.

<57> Section 3.1.5.9.7: This is only implemented by the NTFS file system.

<58> Section 3.1.5.9.8: This operation is only implemented by the ReFS file system.

<59> Section 3.1.5.9.9: This is only implemented by the NTFS file system.

<60> Section 3.1.5.9.9: Several of the fields being set in this section are specific to how the NTFS file system is implemented and are not defined in the Object Stores Abstract Data Model.

<61> Section 3.1.5.9.10: This is only implemented by the NTFS file system.

<62> Section 3.1.5.9.11: This is only implemented by the ReFS and NTFS file systems.

<63> Section 3.1.5.9.18: Support for this FSCTL is only implemented in the FAT file system. The data returned by this FSCTL is incomplete and incorrect on FAT32, and it is unsupported on all other file systems, as specified in [MS-FSCC] section 2.3.35.

<64> Section 3.1.5.9.18: This operation is only supported by the FAT file system.

<65> Section 3.1.5.9.19: This is only implemented by the ReFS and NTFS file systems.

<66> Section 3.1.5.9.20: This is only implemented by the UDFS file system.

<67> Section 3.1.5.9.21: This is only implemented by the UDFS file system.

<68> Section 3.1.5.9.22: This is only implemented by the ReFS and NTFS file systems.

<69> Section 3.1.5.9.23: This file system request is handled by the optional hierarchical storage management (HSM) file system filter. This filter has been deprecated as of Windows Server 2008 and is a server-only feature.

<70> Section 3.1.5.9.24: This is only implemented by the NTFS file system.

<71> Section 3.1.5.9.24: NTFS File Compression can be disabled globally on a system by setting the registry key HKLM\SYSTEM\CurrentControlSet\Control\FileSystem\NtfsDisableCompression to 1 and then rebooting the system to have the change take effect. Compression can be re-enabled by setting this key to zero and rebooting the system.

<72> Section 3.1.5.9.25: This is only implemented by the UDFS file system on media types that require software defect management.

<73> Section 3.1.5.9.26: This is only implemented by the NTFS file system.

<74> Section 3.1.5.9.27: Only ReFS supports integrity.

<75> Section 3.1.5.9.28: This is only implemented by the NTFS file system.

<76> Section 3.1.5.9.28: The file system only updates LastChangeTime if no user has explicitly set LastChangeTime. Some Windows file systems defer setting the LastChangeTime until the handle is closed.

<77> Section 3.1.5.9.29: This is only implemented by the NTFS file system.

<78> Section 3.1.5.9.29: The file system only updates LastChangeTime if no user has explicitly set LastChangeTime. The NTFS and ReFS file systems defer setting the LastChangeTime until the handle is closed.

<79> Section 3.1.5.9.30: This is only implemented by the ReFS and NTFS file systems.

<80> Section 3.1.5.9.30: The file system only updates LastChangeTime if no user has explicitly set LastChangeTime. The NTFS and ReFS file systems defer setting the LastChangeTime until the handle is closed.

<81> Section 3.1.5.9.31: WinPE stands for the Windows Preinstallation Environment. For more information please see: http://technet.microsoft.com/en-us/library/cc766093(WS.10).aspx

<82> Section 3.1.5.9.32: This is only implemented by the NTFS file system.

<83> Section 3.1.5.9.33: This is only implemented by the ReFS and NTFS file systems.

<84> Section 3.1.5.9.34: This is only implemented by the NTFS file system.

<85> Section 3.1.5.9.35: [SIS] (Single Instance Storage) is an optional feature available in the following versions of Windows Server: Windows Storage Server 2003 R2, Standard Edition, Windows Storage Server 2008, and Windows Storage Server 2008 R2. [SIS] is not supported directly by an of the Windows file systems but is implemented as a file system filter. Please refer to the following article for detailed information about [SIS].

<86> Section 3.1.5.9.35: In the Windows environment file system are implemented in kernel mode. If a NULL security context is specified and the originator of the operation is running in kernel mode, a built-in SYSTEM security context is used that grants all access.

<87> Section 3.1.5.9.35: In the Windows environment file system are implemented in kernel mode. If a NULL security context is specified and the originator of the operation is running in kernel mode, a built-in SYSTEM security context is used that grants all access.

<88> Section 3.1.5.9.35: In the Windows environment this is done by creating a new file in what is known as the "SIS Common Store". Reparse points are attached to any file controlled by [SIS] that contains information on how to access the Common Store file that contains the data for this file. Please see the following article about [SIS] for details on how this is implemented.

<89> Section 3.1.5.9.36: This is only implemented by the NTFS file system.

<90> Section 3.1.5.11.5: Only ReFS supports integrity.

<91> Section 3.1.5.11.5: Only ReFS supports integrity.

<92> Section 3.1.5.11.6: Only ReFS supports integrity.

<93> Section 3.1.5.11.6: Only ReFS supports integrity.

<94> Section 3.1.5.11.10: Only NTFS implements EAs.

<95> Section 3.1.5.11.12: Only NTFS implements EAs.

<96> Section 3.1.5.11.21: Available only in ReFS.

<97> Section 3.1.5.11.21: Available only in ReFS.

<98> Section 3.1.5.11.23: If **Open.Mode** contains neither FILE_SYNCHRONOUS_IO_ALERT nor FILE_SYNCHRONOUS_IO_NONALERT, this operation does not return meaningful information in **OutputBuffer.CurrentByteOffset**, because **Open.CurrentByteOffset** is not maintained for any **Open** that does not have either of those flags set.

<99> Section 3.1.5.11.27: This algorithm is only implemented by NTFS and ReFS. The FAT, EXFAT, CDFS, and UDFS file systems always return 1.

<100> Section 3.1.5.12.5: The following table defines what FileSystemAttributes flags, as defined in [MS-FSCC] section 2.5.1, are set by various Windows file systems and why they are set:

	ReFS	NTFS	FAT	EXFAT	UDFS	CDFS
FILE_SUPPORTS_USN_JOURNAL 0x02000000	Always Set	Set if 3.0 format or higher volume				
FILE_SUPPORTS_OPEN_BY_FILE_ID 0x01000000	Always Set	Always Set			Set if volume mounted read-only	Always Set
FILE_SUPPORTS_EXTENDED_ATTRIBUTES 0x00800000		Always Set				
FILE_SUPPORTS_HARD_LINKS 0x00400000		Always Set			Always Set	
FILE_SUPPORTS_TRANSACTIONS 0x00200000		Set if 3.0 format or higher volume				

	ReFS	NTFS	FAT	EXFAT	UDFS	CDFS
FILE_SEQUENTIAL_WRITE_ONCE 0x00100000					Set if volume not mounted read-only	
FILE_READ_ONLY_VOLUME 0x00080000	Set if volume mounted read-only	Set if volume mounted read-only	Set if volume mounted read-only	Set if volume mounted read-only	Set if volume mounted read-only	Always Set
FILE_NAMED_STREAMS 0x00040000		Always Set			Set if 2.0 format or higher	
FILE_SUPPORTS_ENCRYPTION 0x00020000		Set if 3.0 format or higher volume and encryption is enabled on the system				
FILE_SUPPORTS_OBJECT_IDS 0x00010000		Set if 3.0 format or higher volume				
FILE_VOLUME_IS_COMPRESSED 0x00008000						
FILE_SUPPORTS_REMOTE_STORAGE 0x00000100						
FILE_SUPPORTS_REPARSE_POINTS 0x00000080	Always Set	Set if 3.0 format or higher volume				
FILE_SUPPORTS_SPARSE_FILES 0x00000040		Set if 3.0 format or higher volume				
FILE_VOLUME_QUOTAS 0x00000020		Set if 3.0 format or higher volume				
FILE_FILE_COMPRESSION		Set if				

	ReFS	NTFS	FAT	EXFAT	UDFS	CDFS
0x00000010		volume cluster size is 4K or less				
FILE_PERSISTENT_ACLS 0x00000008	Always Set	Always Set				
FILE_UNICODE_ON_DISK 0x00000004	Always Set	Always Set	Always Set	Always Set	Always Set	Set if Joliet Format
FILE_CASE_PRESERVED_NAMES 0x00000002	Always Set	Always Set	Always Set	Always Set	Always Set	
FILE_CASE_SENSITIVE_SEARCH 0x00000001	Always Set	Always Set			Always Set	Always Set

<101> Section 3.1.5.12.5: The following table defines the MaximumComponentNameLength, as defined in [MS-FSCC] section 2.5.1, that is set by each file system:

	ReFS	NTFS	FAT	EXFAT	UDFS	CDFS
MaximumComponentNameLength Value	255	255	255	255	254	110 if Joliet Format 221 otherwise

<102> Section 3.1.5.12.8: ReFS does not implement object IDs.

<103> Section 3.1.5.14.1: The following table describes the maximum file size supported by various Windows File Systems.

	ReFS	NTFS	FAT	EXFAT	UDFS	CDFS
MaximumFileSize	((2^32)-1) * number of clusters	16 TB for Windows 2000, Windows XP, Windows Server 2003, Windows Vista, Windows Server 2008, Windows 7, and Windows Server 2008 R2 (((2^32)-1) * number of clusters) for Windows 8 and Windows Server 2012 The physical format will support 16 exabytes.	4 GB	16 exabytes	8 TB	8 TB

<104> Section 3.1.5.14.4: The following table describes the maximum file size supported by various Windows File Systems.

	ReFS	NTFS	FAT	EXFAT	UDFS	CDFS
MaximumFileSize	((2^32)-1) * number of clusters	16 TB for Windows 2000, Windows XP, Windows Server 2003, Windows Vista, Windows Server 2008, Windows 7, and Windows Server 2008 R2 (((2^32)-1) * number of clusters) for Windows 8 and Windows Server 2012 The physical format will support 16 exabytes.	4 GB	16 exabytes	8 TB	8 TB

<105> Section 3.1.5.14.5: Only NTFS implements EAs.

<106> Section 3.1.5.14.6: Only NTFS supports FileLinkInformation.

<107> Section 3.1.5.14.9: If **Open.Mode** contains neither FILE_SYNCHRONOUS_IO_ALERT nor FILE_SYNCHRONOUS_IO_NONALERT, this operation does not have any meaningful effect, because **Open.CurrentByteOffset** is not used for any **Open** that does not have either of those flags set.

<108> Section 3.1.5.14.11: The file system only updates LastChangeTime if no user has explicitly set LastChangeTime. Some Windows file systems defer setting the LastChangeTime until the handle is closed.

<109> Section 3.1.5.14.13: ReFS does not implement short names.

<110> Section 3.1.5.14.14: ValidDataLength is an internal implementation detail of the NTFS file system and the ReFS file system. It is not a notion that exists in other Windows file systems. ValidDataLength, as defined by NTFS and ReFS, refers to a high-watermark in the file that is considered to be initialized data by a user writing in the region or by the file system writing zeros. Any reads within that value are required to return data from the persistent store. Any reads beyond that value are required to return zeros. There is no API to query ValidDataLength, and the API to set ValidDataLength only allows the value to increase from the existing value.

<111> Section 3.1.5.15.8: Only NTFS implements object IDs.

<112> Section 3.1.5.16: The file system only updates LastChangeTime if no user has explicitly set LastChangeTime. Some Windows file systems defer setting the LastChangeTime until the handle is closed.

<113> Section 3.1.5.19: In Windows file systems, operations are only cancelable if they are blocked and put on a wait queue of some kind. Operations that are actively being processed are not cancelable.

<114> Section 3.1.5.20: The name of the quota file in the Windows environment is:

\$Extend\$Quota:$Q:$INDEX_ALLOCATION

<115> Section 3.1.5.21: The name of the quota file in the Windows environment is:

\$Extend\$Quota:$Q:$INDEX_ALLOCATION

7 Change Tracking

This section identifies changes that were made to the [MS-FSA] protocol document between the December 2011 and March 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
1.2.2 Informative References	Added content for Windows 8 and Windows Server 2012.	Y	Content updated.
3.1.1.1 Per Volume	Minor changes during review of file for existing content.	N	Content updated.
3.1.1.1 Per Volume	Added content for Windows® 8 operating system and Windows Server® 2012 operating system.	Y	Content updated.
3.1.1.1 Per Volume	Added content for Windows 8.	Y	Content updated.
3.1.1.1 Per Volume	66454 Replaced the prescriptive term MAY with "could" in the description of the TunnelCacheEntries attribute.	N	Content updated.
3.1.1.2 Per TunnelCacheEntry	66454 Replaced the prescriptive term MAY with "could" in the description of the FileShortName attribute.	Y	Content updated.
3.1.1.3 Per File	66454 Clarified normative language in the description of the ReparseTag and ReparseData attributes.	N	Content updated.
3.1.1.4	66454	N	Content

Section	Tracking number (if applicable) and description	Major change (Y or N)	Change type
Per Link	Replaced the prescriptive term MAY with "could" in the description of the description of the ShortName attribute.		updated.
3.1.1.5 Per Stream	66454 Replaced the prescriptive term MAY with "could" in the description of the Name attribute and the oplock field.	N	Content updated.
3.1.1.6 Per Open	66454 Replaced the prescriptive term MAY with "could" in the description of the TargetOplockKey and ParentOplockKey attributes.	N	Content updated.
3.1.4.12 Algorithm to Check for an Oplock Break	66454 Clarified normative language in the description of the OpParam input to the algorithm.	N	Content updated.
3.1.5.1 Server Requests an Open of a File	66454 Replaced the prescriptive term MAY with "could" in the description of the TargetOplockKey and UserCertificate values.	N	Content updated.
3.1.5.2 Server Requests a Read	Changed SectorSize ADM element to LogicalBytesPerSector.	Y	Content updated.
3.1.5.3 Server Requests a Write	Changed SectorSize ADM element to LogicalBytesPerSector.	Y	Content updated.
3.1.5.4 Server Requests Closing an Open	66436 Changed descriptive "must" to prescriptive "MUST".	N	Content updated.
3.1.5.5 Server Requests Querying a Directory	66454 Replaced the prescriptive term MAY with "could" in the description of the FileNamePattern value.	N	Content updated.
3.1.5.9.4 FSCTL_FILE_LEVEL_TRIM	66277 Added section.	Y	New content added.
3.1.5.9.5 FSCTL_FILESYSTEM_GET_STATISTICS	66449 Clarified functionality regarding the return of STATUS_BUFFER_OVERFLOW.	Y	Content updated.
3.1.5.9.8 FSCTL_GET_INTEGRITY_INFORMATION	Added section with content for Windows 8 and Windows Server 2012.	Y	New content added.

Section	Tracking number (if applicable) and description	Major change (Y or N)	Change type
3.1.5.9.9 FSCTL_GET_NTFS_VOLUME_DATA	Changed SectorSize ADM ec/Trn nD		

8 Index