[MS-FSCC]:
File System Control Codes

Intellectual Property Rights Notice for Open Specifications Documentation

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

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

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

Revision Summary

Date	Revision History	Revision Class	Comments
03/14/2007	1.0	Major	Updated and revised the technical content.
04/10/2007	1.1	Minor	Updated the technical content.
05/18/2007	2.0	Major	Updated technical content
06/08/2007	2.1	Minor	Updated the technical content.
07/10/2007	2.1.1	Editorial	Revised and edited the technical content.
08/17/2007	2.1.2	Editorial	Revised and edited the technical content.
09/21/2007	3.0	Major	Updated and revised the technical content.
10/26/2007	4.0	Major	Converted the document to unified format and updated the technical content.
01/25/2008	4.0.1	Editorial	Revised and edited the technical content.
03/14/2008	4.0.2	Editorial	Revised and edited the technical content.
06/20/2008	5.0	Major	Updated and revised the technical content.
07/25/2008	6.0	Major	Updated and revised the technical content.
08/29/2008	7.0	Major	Updated and revised the technical content.
10/24/2008	8.0	Major	Updated and revised the technical content.
12/05/2008	9.0	Major	Updated and revised the technical content.
01/16/2009	10.0	Major	Updated and revised the technical content.
02/27/2009	11.0	Major	Updated and revised the technical content.
04/10/2009	12.0	Major	Updated and revised the technical content.
05/22/2009	13.0	Major	Updated and revised the technical content.
07/02/2009	14.0	Major	Updated and revised the technical content.
08/14/2009	15.0	Major	Updated and revised the technical content.
09/25/2009	16.0	Major	Updated and revised the technical content.
11/06/2009	17.0	Major	Updated and revised the technical content.
12/18/2009	18.0	Major	Updated and revised the technical content.
01/29/2010	19.0	Major	Updated and revised the technical content.
03/12/2010	20.0	Major	Updated and revised the technical content.

Date	Revision History	Revision Class	Comments
04/23/2010	21.0	Major	Updated and revised the technical content.
06/04/2010	22.0	Major	Updated and revised the technical content.
07/16/2010	23.0	Major	Significantly changed the technical content.
08/27/2010	24.0	Major	Significantly changed the technical content.
10/08/2010	25.0	Major	Significantly changed the technical content.
11/19/2010	25.1	Minor	Clarified the meaning of the technical content.
01/07/2011	25.1	No change	No changes to the meaning, language, or formatting of the technical content.
02/11/2011	26.0	Major	Significantly changed the technical content.
03/25/2011	27.0	Major	Significantly changed the technical content.
05/06/2011	28.0	Major	Significantly changed the technical content.
06/17/2011	28.1	Minor	Clarified the meaning of the technical content.

Contents

1 Introduction

This specification defines the network format of native Microsoft Windows® structures that may be used within other protocols. It also describes the structure of common Windows native file system control codes, file information levels, and file system information levels that are issued in client/server and server/server communications. These structures do not result in a protocol, but their structure is common across multiple protocols. As such, they are placed in this document as a reference that can be used by other protocols to ensure consistency and accuracy.

1.1 Glossary

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

8.3 name
binary large object (BLOB)
FAT file system
Fid
file stream
FSCTL
globally unique identifier (GUID)
logical cluster number (LCN)
named stream
NetBIOS name
NTFS
object identifier (OID)
partition
replica set
sector
security identifier (SID)
single-instance storage (SIS)
stream
symbolic link
Unicode character
Universal Disk Format (UDF)
update sequence number (USN)
volume

The following terms are specific to this document:

alternate name: An **8.3 name** that can optionally be generated when a file is created. A file will not have an **alternate name** if the user wants to optimize performance, or if the name of the file already uses the 8.3 format.

chunk: The amount of data that the operating system's implementation of the Lempel-Ziv compression algorithm tries to compress at one time. The **compression unit** size used by the file system is always a multiple of the underlying compression algorithm's chunk size. For more information on the Lempel-Ziv compression algorithm, see [UASDC].

cluster: The smallest allocation unit on a **volume**.

compression unit: The amount of data that **NTFS** tries to compress at one time. Compression of large files is accomplished as a series of compressions of data blocks, each at the most compression unit bytes in size.

compression unit shift: The number of bits by which to left-shift a 1 bit to arrive at the **compression unit** size.

content indexing service: A Windows service that extracts content from files and constructs an indexed catalog to facilitate efficient and rapid searching.

disk quota: Maximum amount of data a user may store on a disk **volume**.

Distributed Link Tracking: A Windows service that tracks files as they move across **volumes** on a system and across systems on a network. This Windows service uses **FSCTL**.

dot directory name: In a pathname, a directory name component of "." or "..". For more details, see section 2.1.5.1.

file name component: The portion of a file name between path separator characters (or backslashes).

file record segment: A record in the **master file table** that contains attributes for a specific file on an **NTFS volume**. The file record segment is always 1,024 bytes (1 kilobyte) in size.

filter: Type of driver that is layered between the kernel and a base file system (such as **FAT** or **NTFS**) that receives I/O request packets on their way to and from the base file system. The term **filter** can refer to **legacy filters** or **minifilters**.

filter manager: A file system **filter** driver that simplifies the development of other file system **filter** drivers. Although it is possible to write a filter driver that manages other **filters**, for the purposes of this document, the phrase **filter manager** refers only to the file system **filter manager**, which is an operating system component. A **filter** driver developed to the **filter manager** model is called a **minifilter**.

independent software vendor (ISV): A company or organization that develops software solutions that may utilize this specification.

legacy filter: A file system **filter** that does not work with the Windows file system **filter manager**.

master file table (MFT): On an **NTFS volume**, the MFT is a relational database that consists of rows of file records and columns of file attributes. It contains at least one entry for every file on an **NTFS volume**, including the MFT itself. The MFT stores the information required to retrieve files from the **NTFS** partition.

master file table mirror (MFT2/MFTMirr): On an **NTFS volume**, the MFT2 is a redundant copy of the first four (4) records of the **MFT**.

minifilter: A file system **filter** developed to work with the file system **filter manager**.

object-oriented file system: In the context of file system control codes, a file system that allows the assignment of object IDs to files.

ReparseGuid: A 16-byte **GUID** that uniquely identifies the owner of the **reparse point**. **Reparse point GUIDs** are assigned by the implementer of a file system, the file system **filter** driver, or the **minifilter** driver. The implementer must generate one **GUID** to use with their assigned **reparse point tag**, and must always use this **GUID** as the ReparseGuid for that **tag**.

reparse point: An attribute that can be added to a file to store a collection of user-defined data that is opaque to **NTFS**. If a file that has a reparse point is opened, the open will normally fail

with STATUS_REPARSE, so that the relevant file system **filter** driver can detect the open of a file associated with (owned by) this **reparse point**. At that point, each installed **filter** driver can check to see if it is the owner of the **reparse point**, and, if so, perform any special processing required for a file with that **reparse point**. The format of this data is understood by the application that stores the data and the file system **filter** that interprets the data and processes the file. For example, an encryption **filter** that is marked as the owner of a file's **reparse point** could look up the encryption key for that file. A file can have (at most) 1 **reparse point** associated with it.

reparse point tag: A unique identifier for a file system **filter** driver stored within a file's optional **reparse point** data that indicates the file system **filter** driver that performs additional filter-defined processing on a file during I/O operations. An implementer may request more than one **reparse point** for use with a file system, a file system **filter** driver, or a **minifilter** driver. To request a reparse point tag, use the reparse point tag request form. For more information, see [WHDC-RPTR].

short name: This has the same definition as **alternate name**.

sparse file: A file containing large sections of data composed only of zeros, which is marked as such in the **NTFS**. The file system saves disk space by only allocating as many ranges on disk as are required to completely reconstruct the nonzero data. When an attempt is made to read in the non-allocated portions of the file (also known as holes), the file system automatically returns zeros to the caller.

sub-read and sub-write: An I/O operation sent by the file system to the storage stack that is part of a larger file I/O operation. Sometimes large file reads and writes are broken down by the file system into smaller reads and writes, which are then sent to the storage stack.

tag: Another name for a **reparse point**. For instance, the file system **filter manager** FltTagFile routine sets a **reparse point** on a file. Tag is also used to refer to the field in a **reparse point** that identifies what software component put the **reparse point** there.

virtual cluster number (VCN): The **cluster** number relative to the beginning of the file, directory, or **stream** within a file. The **cluster** describing byte 0 in a file is VCN 0.

MAY, SHOULD, MUST, SHOULD NOT, MUST NOT: These terms (in all caps) are used as specified 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-LSAD] Microsoft Corporation, "Local Security Authority (Domain Policy) Remote Protocol Specification".

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

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

[MS-WDVME] Microsoft Corporation, "Web Distributed Authoring and Versioning (WebDAV) Protocol: Microsoft Extensions".

[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

1.2.2 Informative References

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

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

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

[MS-DLTW] Microsoft Corporation, "Distributed Link Tracking: Workstation Protocol Specification".

[MS-EFSR] Microsoft Corporation, "Encrypting File System Remote (EFSRPC) Protocol Specification".

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

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

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

[MSDN-CJ] Microsoft Corporation, "Change Journals", http://msdn.microsoft.com/en-us/library/aa363798.aspx

[MSFT-NTFS] Microsoft Corporation, "NTFS Technical Reference", March 2003, http://technet2.microsoft.com/WindowsServer/en/Library/81cc8a8a-bd32-4786-a849-03245d68d8e41033.mspx

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

[REPARSE] Microsoft Corporation, "Reparse Points", http://msdn.microsoft.com/en-us/library/aa365503.aspx

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

[SPARSE] Microsoft Corporation, "Sparse Files", http://msdn.microsoft.com/en-us/library/aa365564.aspx

[UASDC] Ziv, J. and Lempel, A., "A Universal Algorithm for Sequential Data Compression", May 1977, http://ieeexplore.ieee.org/iel5/18/22696/01055714.pdf

[WHDC-RPTR] Microsoft Corporation, "Reparse Point Tag Request", August 2002, http://www.microsoft.com/whdc/devtools/ifskit/reparse.mspx

[UDF] Optical Storage Technology Association, "UDF Specification, Revision 2.60", March 2005, http://www.osta.org/specs/pdf/udf260.pdf

1.3 Overview

This document describes the structure of common file system control (**FSCTL**) codes, file information levels, and file system information levels that are issued in client/server and server/server communications. These structures do not result in a protocol, but their structure is common across multiple protocols. As such, they are placed in this document as a reference that can be used by other protocols to ensure consistency and accuracy.

File system control codes are parameters to the device I/O control interface between applications and the operating system. These device I/O control functions, like other I/O functions, accept a file handle as a parameter, indicating the resource on which the requested operation should be performed. When the operating system detects that a handle corresponds to a file on a remote file server, the request may be redirected over the network to the server where the file is stored.

The following topics are addressed in this specification:

- Common file system control operations, including the control code itself and the input/output parameters.
- File information classes and their corresponding structures.
- File system information classes and their corresponding structures.
- File attribute definitions and NTSTATUS code definitions referenced by the file system control code, file information level, and file system information-level documentation.

1.4 Relationship to Protocols and Other Structures

Versions 1 and 2 of the Server Message Block (SMB) Protocol, as specified in [MS-SMB] and [MS-SMB2], rely on the structures and definitions in this document to interpret certain fields that may be sent or received as part of its processing.

1.5 Applicability Statement

The structures and classes defined in this document are useful for any lower-level protocol that serializes and exchanges file information levels, file system information levels, and file system control operations without needing to remap this information into a protocol-specific representation.

1.6 Versioning and Localization

None.

1.7 Vendor-Extensible Fields

File system control codes that are used to set **reparse point** data specify a **ReparseTag** field value that identifies the file system **filter** that understands the application-specific reparse point data format. A vendor developing an application protocol that sets reparse point data should request a unique reparse **tag** for that application from Microsoft by following the instructions described in [WHDC-RPTR]. For more information about reparse points, see [REPARSE].

This protocol uses NTSTATUS values, as specified in [MS-ERREF]. 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.

2 Structures

The structures specified in this document have no transport requirements of their own. Instead, they are packaged and transported in accordance with the protocol that makes use of them, such as the Server Message Block (SMB) Protocol, as specified in [MS-SMB]. A server receiving one of these structures passes the structure to an implementation-defined function that performs the indicated operation on a file, a file system, or a **volume**.

The following sections specify how File System Control Codes messages are encapsulated on the wire and common File System Control Codes data types.

This document references commonly used data types as defined in [MS-DTYP].

Unless otherwise qualified, instances of **GUID** in this section refer to [MS-DTYP] section 2.3.2.

2.1 Common Data Types

2.1.1 Time

Unless otherwise noted, **Time** fields are 64-bit signed integers representing the number of 100-nanosecond intervals that have elapsed since January 1, 1601, Coordinated Universal Time (UTC).

See **FILETIME** ([MS-DTYP] section 2.3.1) for related information.

For information regarding the semantics of the file timestamps of the **CreationTime**, **LastAccessTime**, **LastWriteTime**, and **ChangeTime** fields, see [FSBO] section 6.

2.1.2 Reparse Point Data Structures

For conceptual information about reparse points, see [REPARSE].

2.1.2.1 Reparse Tags

Each reparse point has a **reparse tag**. The reparse tag uniquely identifies the owner of that reparse point. The owner is the implementer of the file system filter driver associated with a reparse tag.

Reparse tags are exposed to clients for third-party applications. Those applications can set, get, and process reparse tags as needed. Third parties MUST request a reserved reparse tag value to ensure that conflicting tag values do not occur. [WHDC-RPTR]<1>

The following reparse tags, with the exception of IO_REPARSE_TAG_SYMLINK, are processed on the server and are not processed by a client after transmission over the wire. Clients should treat associated reparse data as opaque data. <2>

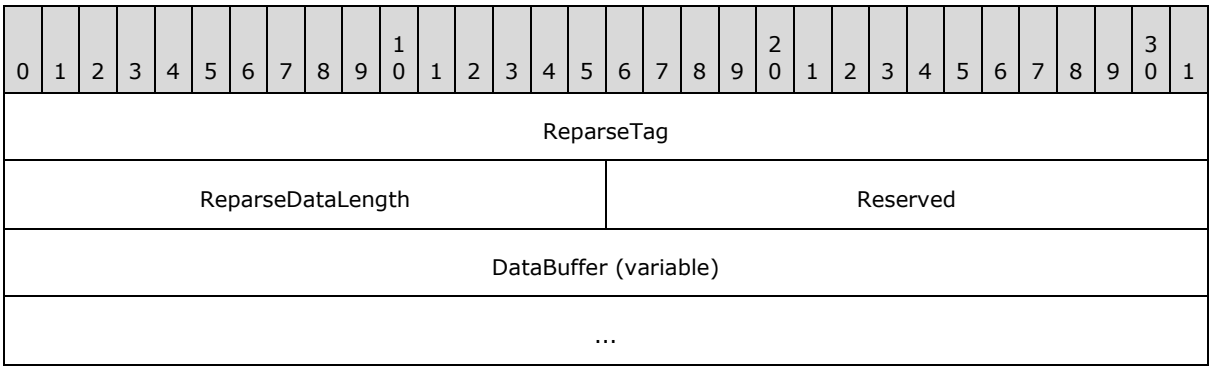
Value	Meaning
IO_REPARSE_TAG_RESERVED_ZERO 0x00000000	Reserved reparse tag value.
IO_REPARSE_TAG_RESERVED_ONE 0x00000001	Reserved reparse tag value.
IO_REPARSE_TAG_MOUNT_POINT 0xA0000003	Used for mount point support, specified in section 2.1.2.5.

Value	Meaning
IO_REPARSE_TAG_HSM 0xC0000004	Obsolete. Used by legacy Hierarchical Storage Manager Product.
IO_REPARSE_TAG_HSM2 0x80000006	Obsolete. Used by legacy Hierarchical Storage Manager Product.
IO_REPARSE_TAG_DRIVER_EXTENDER 0x80000005	Home server drive extender.<3>.
IO_REPARSE_TAG_SIS 0x80000007	Used by **single-instance storage (SIS)** filter driver. Server-side interpretation only, not meaningful over the wire.
IO_REPARSE_TAG_DFS 0x8000000A	Used by the DFS filter. The DFS is described in the Distributed File System (DFS): Referral Protocol Specification [MS-DFSC]. Server-side interpretation only, not meaningful over the wire.
IO_REPARSE_TAG_DFSR 0x80000012	Used by the DFS filter. The DFS is described in [MS-DFSC]. Server-side interpretation only, not meaningful over the wire.
IO_REPARSE_TAG_FILTER_MANAGER 0x8000000B	Used by **filter manager** test harness.<4>
IO_REPARSE_TAG_SYMLINK 0xA000000C	Used for **symbolic link** support. See section 2.1.2.4.

2.1.2.2 REPARSE_DATA_BUFFER

The REPARSE_DATA_BUFFER data element stores data for a reparse point. This reparse data buffer MUST be used only with reparse tag values whose high bit is set to 1.

This data element has two subtypes: Symbolic Link Reparse Data Buffer (section 2.1.2.4) and Mount Point Reparse Data Buffer (section 2.1.2.5).



ReparseTag (4 bytes): A 32-bit unsigned integer value containing the reparse point tag that uniquely identifies the owner of the reparse point.

ReparseDataLength (2 bytes): A 16-bit unsigned integer value containing the size, in bytes, of the reparse data in the **DataBuffer** member.

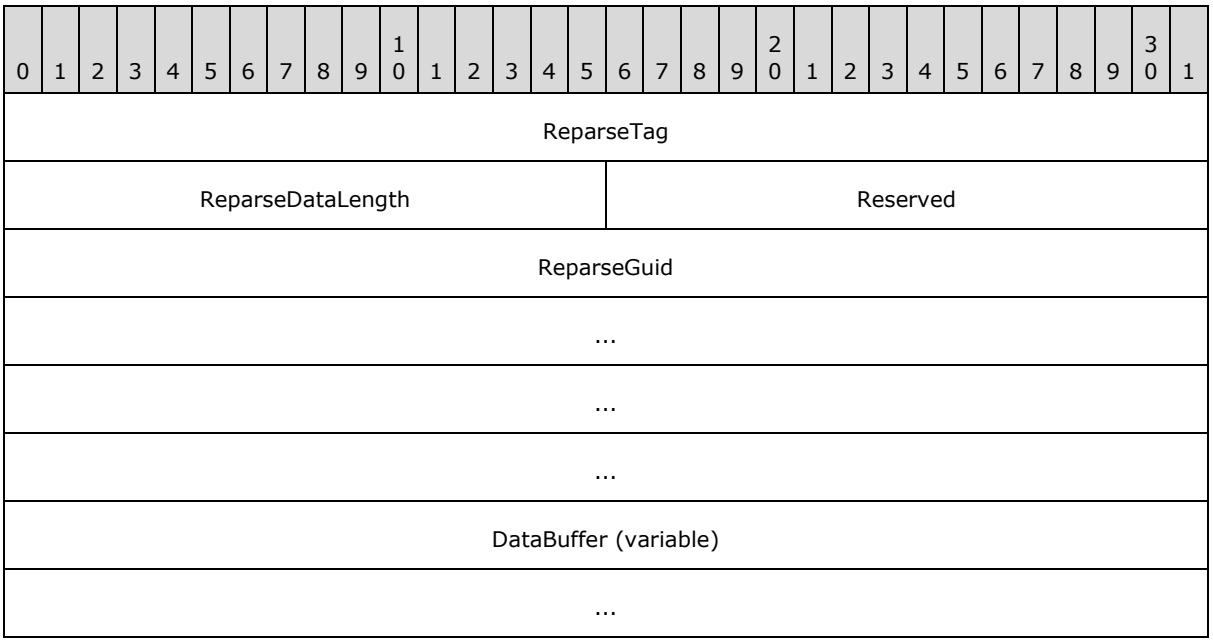
Reserved (2 bytes): A 16-bit field. This field is reserved. This field SHOULD be set to 0, and MUST be ignored.

DataBuffer (variable): A variable-length array of 8-bit unsigned integer values containing reparse-specific data for the reparse point. The format of this data is defined by the owner (that is, the implementer of the filter driver associated with the specified ReparseTag) of the reparse point.

2.1.2.3 REPARSE_GUID_DATA_BUFFER

The REPARSE_GUID_DATA_BUFFER data element stores data for a reparse point and associates a GUID with the reparse tag. This reparse data buffer MUST be used only with reparse tag values whose high bit is set to 0.

Reparse point **GUIDs** are assigned by the **independent software vendor (ISV)**. An ISV MUST link one GUID to each assigned reparse point tag, and MUST always use that GUID with that tag.



ReparseTag (4 bytes): A 32-bit unsigned integer value containing the reparse point tag that uniquely identifies the owner of the reparse point.

ReparseDataLength (2 bytes): A 16-bit unsigned integer value containing the size, in bytes, of the reparse data in the **DataBuffer** member.

Reserved (2 bytes): A 16-bit field. This field SHOULD be set to 0 by the client, and MUST be ignored by the server.

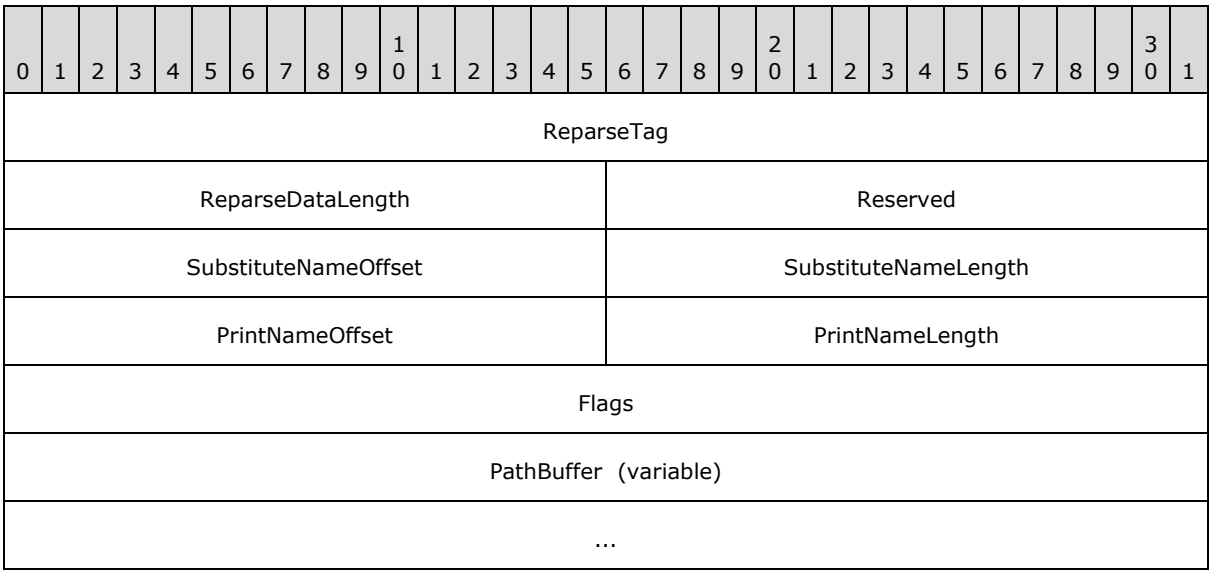
ReparseGuid (16 bytes): A 16-byte GUID that uniquely identifies the owner of the reparse point. Reparse point GUIDs are not assigned by Microsoft. A reparse point implementer MUST select one GUID to be used with their assigned reparse point tag to uniquely identify that reparse point. For more information, see [REPARSE].

DataBuffer (variable): The content of this buffer is opaque to the file system. On receipt, its content MUST be preserved and properly returned to the caller.

2.1.2.4 Symbolic Link Reparse Data Buffer

The Symbolic Link Reparse Data Buffer data element is a subtype of REPARSE_DATA_BUFFER, which contains information on symbolic link reparse points. This reparse data buffer MUST be used only with reparse tag values whose high bit is set to 1.

A symbolic link has a substitute name and a print name associated with it. The substitute name is a pathname (section 2.1.5) identifying the target of the symbolic link. The print name SHOULD be an informative pathname, suitable for display to a user, that also identifies the target of the symbolic link. Either pathname can contain dot directory names as specified in section 2.1.5.1.



ReparseTag (4 bytes): A 32-bit unsigned integer value containing the reparse point tag that uniquely identifies the owner (that is, the implementer of the filter driver associated with this ReparseTag) of the reparse point. This value MUST be 0xA000000C.

ReparseDataLength (2 bytes): A 16-bit unsigned integer value containing the size, in bytes, of the reparse data that follows the common portion of the REPARSE_DATA_BUFFER element. This value is the length of the data starting at the **SubstituteNameOffset** field (or the size of the **PathBuffer** field, in bytes, plus 12).

Reserved (2 bytes): A 16-bit field. This field is not used. It SHOULD be set to 0 and MUST be ignored.

SubstituteNameOffset (2 bytes): A 16-bit unsigned integer that contains the offset, in bytes, of the substitute name string in the **PathBuffer** array, computed as an offset from byte 0 of **PathBuffer**. Note that this offset must be divided by 2 to get the array index.

SubstituteNameLength (2 bytes): A 16-bit unsigned integer that contains the length, in bytes, of the substitute name string. If this string is null-terminated, **SubstituteNameLength** does not include the Unicode null character.

PrintNameOffset (2 bytes): A 16-bit unsigned integer that contains the offset, in bytes, of the print name string in the **PathBuffer** array, computed as an offset from byte 0 of **PathBuffer**. Note that this offset must be divided by 2 to get the array index.

PrintNameLength (2 bytes): A 16-bit unsigned integer that contains the length, in bytes, of the print name string. If this string is null-terminated, **PrintNameLength** does not include the Unicode null character.

Flags (4 bytes): A 32-bit field that specifies whether the substitute name is a full path name or a path name relative to the directory containing the symbolic link.

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

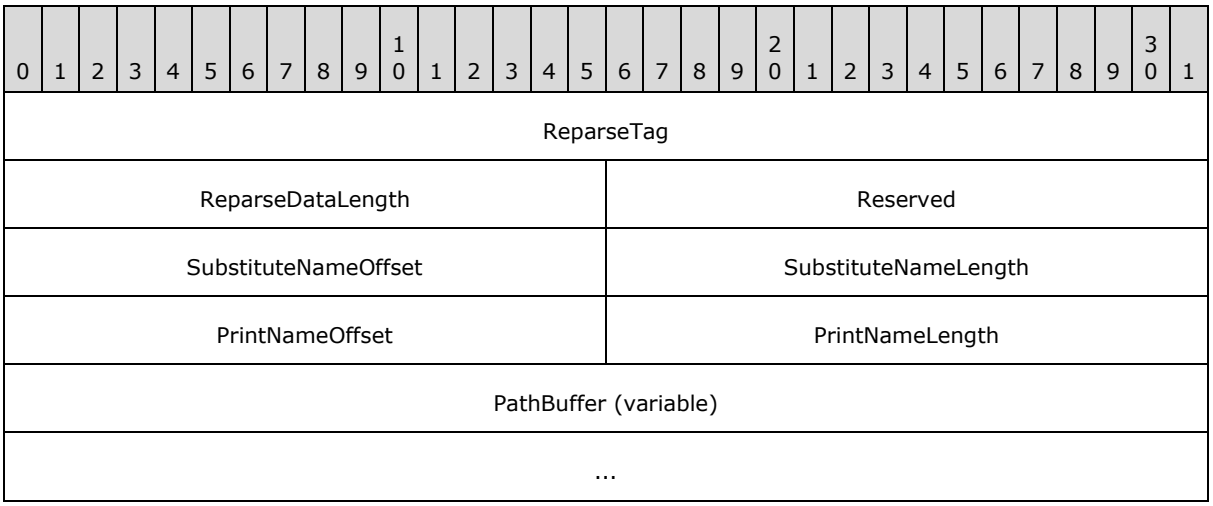
Value	Meaning
0x00000000	The substitute name is a full path name.
SYMLINK_FLAG_RELATIVE 0x00000001	The substitute name is a path name relative to the directory containing the symbolic link.

PathBuffer (variable): Unicode character array that contains the substitute name string and print name string. The substitute name and print name strings can appear in any order in the **PathBuffer**. To locate the substitute name and print name strings in the **PathBuffer**, use the **SubstituteNameOffset**, **SubstituteNameLength**, **PrintNameOffset**, and **PrintNameLength** members.

2.1.2.5 Mount Point Reparse Data Buffer

The Mount Point Reparse Data Buffer data element is a subtype of REPARSE_DATA_BUFFER, which contains information about mount point reparse points. This reparse data buffer MUST be used only with reparse tag values whose high bit is set to 1.

A mount point has a substitute name and a print name associated with it. The substitute name is a pathname (section 2.1.5) identifying the target of the mount point. The print name SHOULD be an informative pathname (section 2.1.5), suitable for display to a user, that also identifies the target of the mount point. Neither of these pathnames can contain dot directory names.



ReparseTag (4 bytes): A 32-bit unsigned integer value containing the reparse point tag that uniquely identifies the owner (that is, the implementer of the filter driver associated with this ReparseTag) of the reparse point. This value MUST be 0xA0000003.

ReparseDataLength (2 bytes): A 16-bit unsigned integer value containing the size, in bytes, of the reparse data that follows the common portion of the REPARSE_DATA_BUFFER element. This value is the length of the data starting at the **SubstituteNameOffset** field (or the size of the **PathBuffer** field, in bytes, plus 8).

Reserved (2 bytes): A 16-bit field. This field is not used. It SHOULD be set to 0, and MUST be ignored.

SubstituteNameOffset (2 bytes): A 16-bit unsigned integer that contains the offset, in bytes, of the substitute name string in the **PathBuffer** array, computed as an offset from byte 0 of **PathBuffer**. Note that this offset must be divided by 2 to get the array index.

SubstituteNameLength (2 bytes): A 16-bit unsigned integer that contains the length, in bytes, of the substitute name string. If this string is null-terminated, **SubstituteNameLength** does not include the Unicode null character.

PrintNameOffset (2 bytes): A 16-bit unsigned integer that contains the offset, in bytes, of the print name string in the **PathBuffer** array, computed as an offset from byte 0 of **PathBuffer**. Note that this offset must be divided by 2 to get the array index.

PrintNameLength (2 bytes): A 16-bit unsigned integer that contains the length, in bytes, of the print name string. If this string is null-terminated, **PrintNameLength** does not include the Unicode null character.

PathBuffer (variable): Unicode character array that contains the substitute name string and print name string. The substitute name and print name strings can appear in any order in **PathBuffer**. To locate the substitute name and print name strings in the **PathBuffer** field, use the **SubstituteNameOffset**, **SubstituteNameLength**, **PrintNameOffset**, and **PrintNameLength** members.

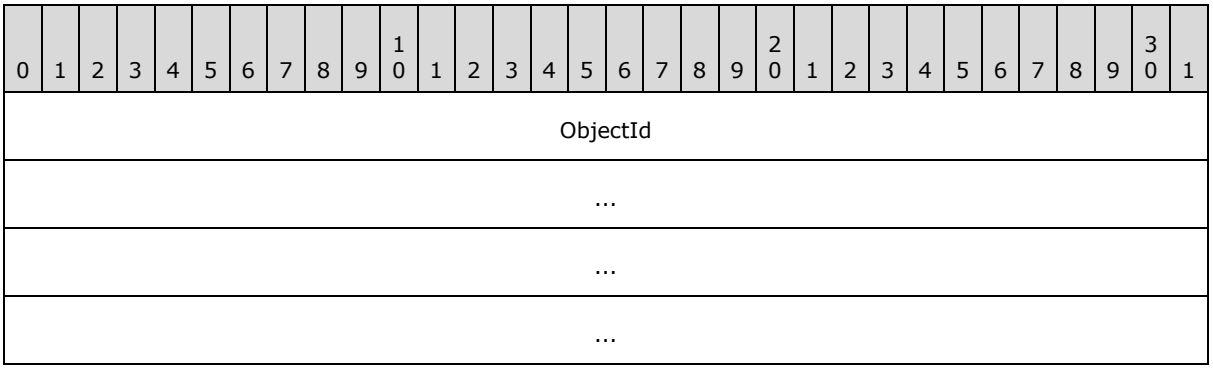
2.1.3 FILE_OBJECTID_BUFFER Structure

The FILE_OBJECTID_BUFFER structure contains extended metadata for a file system object, including its object ID. This data element MUST be in one of the following two formats:

- FILE_OBJECTID_BUFFER Type 1
- FILE_OBJECTID_BUFFER Type 2

2.1.3.1 FILE_OBJECTID_BUFFER Type 1

The first possible structure for the FILE_OBJECTID_BUFFER data element is as follows.



BirthVolumeId
...
...
...
BirthObjectId
...
...
...
DomainId
...
...
...

ObjectId (16 bytes): A 16-byte GUID that uniquely identifies the file or directory within the volume on which it resides. Specifically, the same object ID can be assigned to another file or directory on a different volume, but it MUST NOT be assigned to another file or directory on the same volume.

BirthVolumeId (16 bytes): A 16-byte 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 object identifier of the volume on which the object presently resides.

BirthObjectId (16 bytes): A 16-byte GUID value containing the object identifier of the object at the time it was created. Copy operations, move operations, or other file operations MAY change the value of the **ObjectId** member. Therefore, the BirthObjectId is potentially different from the **ObjectId** member at present. Specifically, the same object ID MAY be assigned to another file or directory on a different volume, but it MUST NOT be assigned to another file or directory on the same volume. The object ID is assigned at file creation time.<5>

DomainId (16 bytes): A 16-byte GUID value containing the domain identifier. This value is unused; it SHOULD be zero, and MUST be ignored.<6>

2.1.3.2 FILE_OBJECTID_BUFFER Type 2

The second possible structure for the FILE_OBJECTID_BUFFER data element is as follows.

0	1	2	3	4	5	6	7	8	9	10	1	2	3	4	5	6	7	8	9	20	1	2	3	4	5	6	7	8	9	30	1
ObjectId																															
...																															
...																															
...																															
ExtendedInfo																															
...																															
...																															
...																															
...																															
...																															
...																															
...																															
(ExtendedInfo cont'd for 4 rows)																															

ObjectId (16 bytes): A 16-byte GUID that uniquely identifies the file or directory within the volume on which it resides. Specifically, the same object ID can be assigned to another file or directory on a different volume, but it MUST NOT be assigned to another file or directory on the same volume.

ExtendedInfo (48 bytes): A 48-byte value containing extended data that was set with the FSCTL_SET_OBJECT_ID_EXTENDED request. This field contains application-specific data.<7>

2.1.4 Alternate Data Streams

A file system MAY<8> support alternate data streams within a file or a directory. For a general description of **file streams**, see [MS-GLOS].

Every file has a default stream, which is the stream that is referenced when no stream name component is specified as part of the pathname. A directory does not have a default data stream; however, it can have named alternate data streams.

For more information on stream naming, see section 2.1.5; for more information on streams in general, see [MS-WSO], section 11.

2.1.5 Pathname

A pathname has the following characteristics:

- A pathname MUST be no more than 32,760 characters in length.
- A pathname is composed of one or more pathname components separated by the "\" backslash character. All pathname components other than the last pathname component denote directories or reparse points. The last pathname component denotes a directory, a file, a stream, or a reparse point.
- A leading "\" backslash character is optional, and determines whether a pathname is absolute or relative:
  - A pathname that begins with a leading "\" backslash character, for example, "\a\b\c", is an absolute pathname. An absolute pathname SHOULD be evaluated relative to the root directory.
  - A pathname that omits a leading "\" backslash character, for example, "a\b\c", is a relative pathname. A relative pathname MAY be evaluated relative to any directory, such as an application's current working directory.
- Each pathname component has one of the following forms:
  - A **dot directory name** as specified in section 2.1.5.1.
  - A filename as specified in section 2.1.5.2, optionally followed by a ":" colon character and a streamname as specified in section 2.1.5.3, optionally followed by a ":" colon character and a streamtype as specified in section 2.1.5.4. The streamname, if specified, MAY be zero-length only if streamtype is also specified; otherwise, it MUST be at least one character. The streamtype, if specified, MUST be at least one character.
- Each pathname component MUST be no more than 255 characters in length.

2.1.5.1 Dot Directory Names

The pathname components of "." (single period) and ".." (two periods) are reserved as dot directory names.

Except where explicitly permitted, a pathname component that is a dot directory name MUST NOT be sent over the wire.

When parsing pathname components, a dot directory name of "." refers to the current directory name component and a dot directory name of ".." refers to the parent directory name of the current directory name component.

Some examples to illustrate:

- In the pathname "dirA\.\dirB", the "." refers to dirA, so this expression is equivalent to "dirA\dirB".
- In the pathname "dirA\dirB\..\dirC", the ".." refers to dirA, so this expression is equivalent to "dirA\dirC".

A dot directory name of ".." at the root of a share MUST be treated as equivalent to ".". For example: \\ServerX\ShareY\..\dirA is equivalent to \\ServerX\ShareY\.\dirA (which is equivalent to \\ServerX\ShareY\dirA).

2.1.5.2 Filename

- All **Unicode characters** are legal in a filename except the following:
  - The characters " \ / [ ] : | < > + = ; , * ?
  - Control characters, ranging from 0x00 through 0x1F.

2.1.5.2.1 8.3 Filename

An 8.3 filename (also referred to as a DOS name, a **short name**, or an 8.3-compliant filename) is a filename that conforms to the following restrictions:

- An 8.3 filename MUST only contain characters that can be represented in ASCII, in the range below 0x80.
- An 8.3 filename MUST NOT contain the " " space character.
- An 8.3 filename MUST NOT contain more than one "." period character.
- The general form of a valid 8.3 filename is a base filename, optionally followed by the "." period character and a filename extension.
  - The base filename MUST be 1-8 characters in length and MUST NOT contain a "." period character.
  - The filename extension, if present, MUST be 1-3 characters in length and MUST NOT contain a "." period character.

2.1.5.3 Streamname

- All Unicode characters are legal in a streamname component except the following:
  - The characters \ / :
  - Control character 0x00.
- A zero-length streamname denotes the default **stream**.

2.1.5.4 Streamtype

- All Unicode characters are legal in a streamtype component except the following:
  - The characters \ / :
  - Control character 0x00.

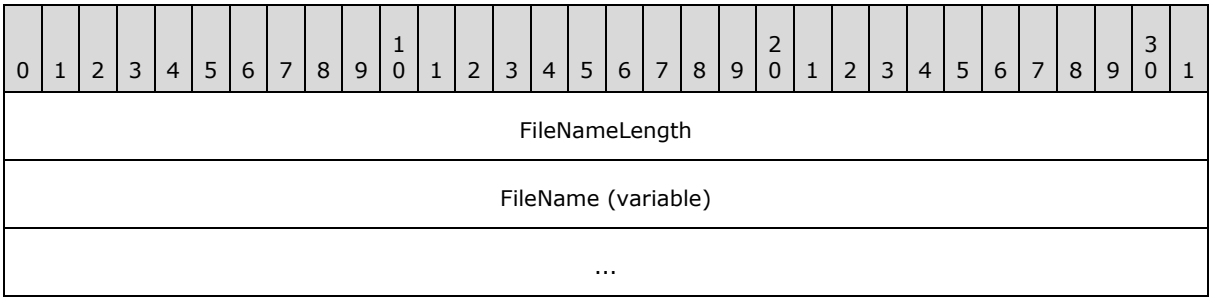
2.1.6 Share name

A share name has the following characteristics:

- A share name MUST be no more than 80 characters in length.
- The following characters are illegal in a share name: "\/[]:|<>+=;,?
- Control characters in range 0x00 through 0x1F, inclusive, are illegal in a share name.
- All other Unicode characters are legal.

2.1.7 FILE_NAME_INFORMATION

The FILE_NAME_INFORMATION data element is as follows.



FileNameLength (4 bytes): A 32-bit unsigned integer that specifies the length, in bytes, of the file name contained within the **FileName** field.

FileName (variable): A sequence of Unicode characters containing the file name. The name string is not null-terminated. There are scenarios where one or more padding characters may be at the end of the string due to buffer alignment requirements, but their presence and their values should not be relied upon. When working with this field, use **FileNameLength** to determine the length of the file name rather than assuming the presence of a trailing null delimiter. This field MUST NOT begin with a path separator character (backslash).

2.1.8 Boolean

A Boolean data type is a primitive that has one of two possible values: TRUE and FALSE, which are defined as follows:

TRUE: A sender MUST use any nonzero value to denote a TRUE. A receiver MUST interpret any nonzero value as TRUE. <9>

FALSE: A sender MUST use a zero value to denote a FALSE. A receiver MUST interpret a zero value as FALSE.

2.2 Status Codes

This specification uses NTSTATUS status codes, as specified in [MS-ERREF] section 2.3. The format of a status code MUST be as specified in [MS-ERREF].

The reply message for each FSCTL lists the error codes that are directly generated by the function that implements the specified FSCTL. Error codes also may be generated by code below the file system (such as RAID drivers or disk drivers) or above the file system (such as virus scanners).

A server SHOULD return a status of STATUS_INVALID_DEVICE_REQUEST when an FSCTL is not supported remotely or is not supported on the file system on which the file or directory handle specified by the FSCTL exists.<10><11>

STATUS_BUFFER_OVERFLOW is a warning code and not an error code. This warning means that the given output buffer is not large enough to contain all of the requested information. Unless otherwise noted, a given operation SHOULD attempt to return as much data as it reasonably can.

2.3 FSCTL Structures

A process invokes an FSCTL on a handle to perform an action against the file or directory associated with the handle. When a server receives an FSCTL request, it SHOULD use the information in the request, which includes a handle and, optionally, an input data buffer, to perform the requested action. How a server performs the action requested by an FSCTL is implementation-dependent.<12>

The following table specifies the system-defined generic FSCTLs that are permitted to be invoked across the network. Generic FSCTLs are used by the local file systems or by multiple components within the system. Any application, service, or driver may define private FSCTLs. Most private FSCTLs are used locally in the internal driver stacks and do not flow over the wire. However, if a component allows its private FSCTLs to flow over the wire, that component is responsible for ensuring the FSCTLs and associated data structures are documented. Examples of such private FSCTLs can be found in [MS-SMB2] and [MS-DFSC].

FSCTL name	FSCTL function number
FSCTL_CREATE_OR_GET_OBJECT_ID	0x900c0
FSCTL_DELETE_OBJECT_ID	0x900a0
FSCTL_DELETE_REPARSE_POINT	0x900ac
FSCTL_FILESYSTEM_GET_STATISTICS	0x90060
FSCTL_FIND_FILES_BY_SID	0x9008f
FSCTL_GET_COMPRESSION	0x9003c
FSCTL_GET_NTFS_VOLUME_DATA	0x90064
FSCTL_GET_OBJECT_ID	0x9009c
FSCTL_GET_REPARSE_POINT	0x900a8
FSCTL_GET_RETRIEVAL_POINTERS	0x90073
FSCTL_IS_PATHNAME_VALID	0x9002c
FSCTL_LMR_SET_LINK_TRACKING_INFORMATION	0x1400ec
FSCTL_PIPE_PEEK	0x11400c
FSCTL_PIPE_TRANSCEIVE	0x11c017
FSCTL_PIPE_WAIT	0x110018
FSCTL_QUERY_FAT_BPB	0x90058
FSCTL_QUERY_ALLOCATED_RANGES	0x940cf
FSCTL_QUERY_ON_DISK_VOLUME_INFO	0x9013c
FSCTL_QUERY_SPARING_INFO	0x90138
FSCTL_READ_FILE_USN_DATA	0x900eb
FSCTL_RECALL_FILE	0x90117

FSCTL name	FSCTL function number
FSCTL_SET_COMPRESSION	0x9c040
FSCTL_SET_DEFECT_MANAGEMENT	0x98134
FSCTL_SET_ENCRYPTION	0x900D7
FSCTL_SET_OBJECT_ID	0x90098
FSCTL_SET_OBJECT_ID_EXTENDED	0x900bc
FSCTL_SET_REPARSE_POINT	0x900a4
FSCTL_SET_SPARSE	0x900c4
FSCTL_SET_ZERO_DATA	0x980c8
FSCTL_SET_ZERO_ON_DEALLOCATION	0x90194
FSCTL_SIS_COPYFILE	0x90100
FSCTL_WRITE_USN_CLOSE_RECORD	0x900ef

2.3.1 FSCTL_CREATE_OR_GET_OBJECT_ID Request

This message requests that the server return the object identifier for the file or directory associated with the handle on which this FSCTL was invoked. If no object identifier exists, the server MUST create one.

This message does not contain any additional data elements.

2.3.2 FSCTL_CREATE_OR_GET_OBJECT_ID Reply

This message returns the results of the FSCTL_CREATE_OR_GET_OBJECT_ID request in a FILE_OBJECTID_BUFFER (section 2.1.3).

This message also returns a status code, as specified in [MS-ERREF] section 2.3. The buffer can be either Type 1 or Type 2 as follows:

- If neither FSCTL_SET_OBJECT_ID_EXTENDED nor FSCTL_SET_OBJECT_ID has been previously issued on the file, then the buffer is of Type 1 and contains implementation-generated values as specified in section 2.1.3.1.
- If FSCTL_SET_OBJECT_ID was used to set the **object ID**, then the buffer is of the type that was used during that FSCTL_SET_OBJECT_ID call.
- If FSCTL_SET_OBJECT_ID_EXTENDED was issued to change the object ID's extended information, then the buffer is of Type 2.

There is no way for the issuer of this FSCTL to determine the returned buffer type without knowing whether the object ID was previously set or modified and by what means (FSCTL_SET_OBJECT_ID_EXTENDED or FSCTL_SET_OBJECT_ID).

The status code returned directly by the function that processes this FSCTL MUST be STATUS_SUCCESS or one of the following.

Error code	Meaning
STATUS_DUPLICATE_NAME 0xC00000BD	The file has no object ID yet, and the file system is unable to generate a unique (to this volume) ID.<13>
STATUS_INVALID_PARAMETER 0xC000000D	The handle is not to a file or directory, or the output buffer is not large enough to contain a FILE_OBJECTID_BUFFER structure.
STATUS_MEDIA_WRITE_PROTECTED 0xC00000A2	The volume is write-protected and changes to it cannot be made. This error code is returned even if the file already has an object ID assigned to it.
STATUS_INVALID_DEVICE_REQUEST 0xC0000010	The file system does not support the use of object IDs.

2.3.3 FSCTL_DELETE_OBJECT_ID Request

This message requests that the server remove the object identifier from the file or directory associated with the handle on which this FSCTL was invoked. The underlying object MUST NOT be deleted. If the file or directory has no object identifier, the request MUST be considered successful.

This message does not contain any additional data elements.

2.3.4 FSCTL_DELETE_OBJECT_ID Reply

This message returns the results of the FSCTL_DELETE_OBJECT_ID request.

The only data item this message returns is a status code, as specified in [MS-ERREF] section 2.3. The status code returned directly by the function that processes this FSCTL MUST be STATUS_SUCCESS or one of the following.

Error code	Meaning
STATUS_ACCESS_DENIED 0xC0000022	The handle was not opened with write access or write attributes access.
STATUS_OBJECT_NAME_NOT_FOUND 0xC0000034	The file or directory has no object ID. This status is not returned on a healthy volume but can be returned if the volume is corrupt.
STATUS_MEDIA_WRITE_PROTECTED 0xC00000A2	The volume is write-protected and changes to it cannot be made.
STATUS_INVALID_DEVICE_REQUEST 0xC0000010	The file system does not support the use of object IDs.

2.3.5 FSCTL_DELETE_REPARSE_POINT Request

This message requests that the server delete the reparse point from the file or directory associated with the handle on which this FSCTL was invoked. The underlying file or directory MUST NOT be deleted.

The message MUST contain a REPARSE_GUID_DATA_BUFFER or a REPARSE_DATA_BUFFER (including subtypes) data element. Both the REPARSE_GUID_DATA_BUFFER and the REPARSE_DATA_BUFFER structures begin with a **ReparseTag** field. The ReparseTag value uniquely

identifies the filter driver that creates/uses the reparse point, and the application's filter driver processes the reparse point data as either a REPARSE_GUID_DATA_BUFFER or a REPARSE_DATA_BUFFER, depending on the structure implemented by the filter driver for that type of reparse point.

This message MUST only be sent for a file or directory handle.

2.3.6 FSCTL_DELETE_REPARSE_POINT Reply

This message returns the result of the FSCTL_DELETE_REPARSE_POINT request.

The only data item this message returns is a status code, as specified in [MS-ERREF] section 2.3. The status code returned directly by the function that processes this FSCTL MUST be STATUS_SUCCESS or one of the following.

Error code	Meaning
STATUS_INVALID_PARAMETER 0xC000000D	A nonzero value was passed for the output buffer's length, or the handle is not to a file or directory.
STATUS_ACCESS_DENIED 0xC0000022	The handle was not opened to write file data or file attributes.
STATUS_IO_REPARSE_DATA_INVALID 0xC0000278	The input buffer's length is neither the size of a REPARSE_DATA_BUFFER nor a REPARSE_GUID_DATA_BUFFER; or the reparse data length is nonzero; or the reparse tag is a third party reparse tag, and the length is other than the size of REPARSE_GUID_DATA_BUFFER.
STATUS_IO_REPARSE_TAG_INVALID 0xC0000276	The specified reparse tag with a value of 0 or 1 is reserved for use by the system and cannot be deleted.
STATUS_NOT_A_REPARSE_POINT 0xC0000275	The file or directory does not have a reparse point.
STATUS_IO_REPARSE_TAG_MISMATCH 0xC0000277	The file or directory has a reparse point but not one with the reparse tag that was specified in this call.
STATUS_REPARSE_ATTRIBUTE_CONFLICT 0xC00002B2	The file or directory has a third party tag, and the Reparse GUID provided does not match the one in the reparse point for this file or directory.

2.3.7 FSCTL_FILESYSTEM_GET_STATISTICS Request

This message requests that the server return the statistical information of the file system such as Type, Version, and so on, as specified in FSCTL_FILESYSTEM_GET_STATISTICS reply, for the file or directory associated with the handle on which this FSCTL was invoked.<14>

This message does not contain any additional data elements.

2.3.8 FSCTL_FILESYSTEM_GET_STATISTICS Reply

This message returns the result of the FSCTL_FILESYSTEM_GET_STATISTICS request message as a pair of structures: a generic structure, FILESYSTEM_STATISTICS, followed by a file system type specific structure that can be either NTFS_STATISTICS, FAT_STATISTICS, or EXFAT_STATISTICS,

depending on the underlying file system type. There is one pair of these structures for each processor.<15>

These statistics contain information about both user and metadata files. User files are available for the user. Metadata files are system files that contain information that the file system uses for its internal organization.

The statistics structures contain fields that may overflow during the server's lifetime. This is by design. When an overflow occurs, the value just wraps. For example 0xfffff000 + 0x2000 will result in 0x1000.

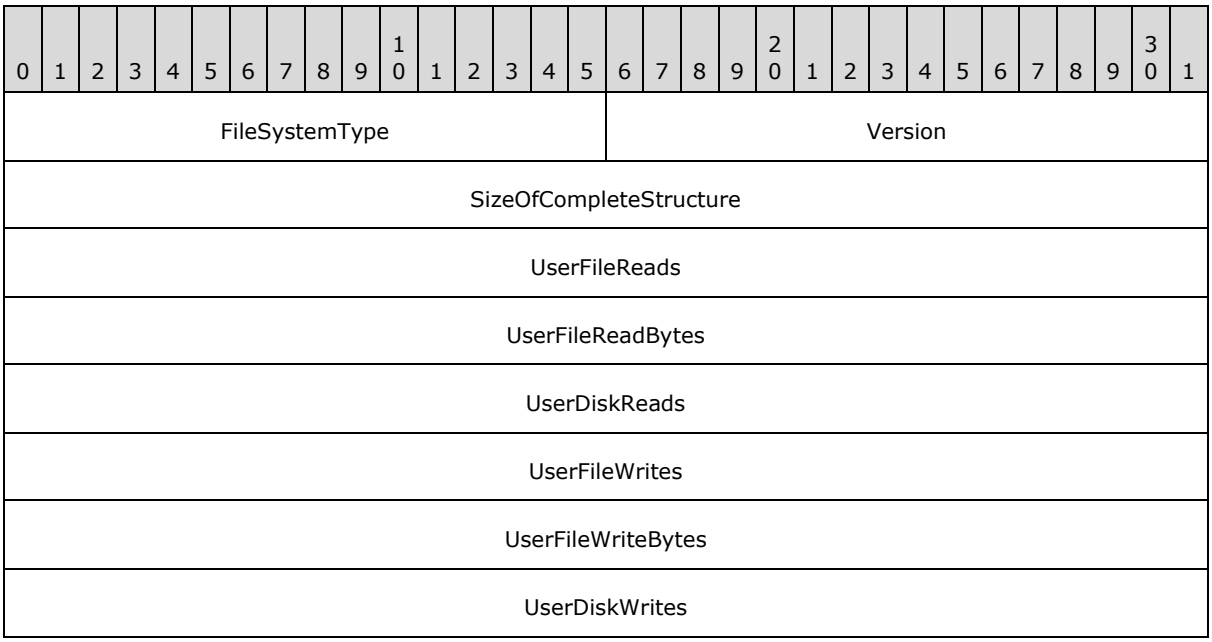
The structures within the output buffer MUST all start on 64-byte boundaries. The final output MUST be padded to a 64-byte boundary. Any padding bytes MUST be filled with zeros.

This message also returns a status code, as specified in [MS-ERREF] section 2.3. The status code returned directly by the function that processes this FSCTL MUST be STATUS_SUCCESS or one of the following.

Error code	Meaning
STATUS_BUFFER_TOO_SMALL 0xC0000023	The output buffer is too small to contain a FILESYSTEM_STATISTICS structure.
STATUS_BUFFER_OVERFLOW 0x80000005	The output buffer was filled before all the statistics data could be returned.

2.3.8.1 FILESYSTEM_STATISTICS

The FILESYSTEM_STATISTICS data element is returned with a FSCTL_FILESYSTEM_GET_STATISTICS reply message. It contains the generic information for the message. The FILESYSTEM_STATISTICS data element is as follows:



MetaDataReads
MetaDataReadBytes
MetaDataDiskReads
MetaDataWrites
MetaDataWriteBytes
MetaDataDiskWrites

FileSystemType (2 bytes): A 16-bit unsigned integer value containing the type of file system. This field MUST contain one of the following values.

Value	Meaning
FILESYSTEM_STATISTICS_TYPE_NTFS 0x0001	The file system is an **NTFS** file system. If this value is set, this structure is followed by an NTFS_STATISTICS structure.
FILESYSTEM_STATISTICS_TYPE_FAT 0x0002	The file system is a **FAT file system**. If this value is set, this structure is followed by a FAT_STATISTICS structure.
FILESYSTEM_STATISTICS_TYPE_EXFAT 0x0003	The file system is an exFAT file system. If this value is set, this structure is followed by an EXFAT_STATISTICS structure.

Version (2 bytes): A 16-bit unsigned integer value containing the version. This field MUST contain 0x00000001.

SizeOfCompleteStructure (4 bytes): A 32-bit unsigned integer value that indicates the size, in bytes, of this structure plus the size of the file system-specific structure that follows this structure, each rounded up to a multiple of 64, then the sum is multiplied by the number of processors. For example, if the size of FILESYSTEM_STATISTICS is 0x38, the size of NTFS_STATISTICS is 0xd4, and there are two processors, the size of the buffer allocated must be 0x280. This is the sum of the sizes of the NTFS_STATISTICS structure and the FILESYSTEM_STATISTICS structure, both rounded up to a multiple of 64 (0x40 + 0x100 = 0x140), and multiplied by the number of processors.

UserFileReads (4 bytes): A 32-bit unsigned integer value containing the number of read operations on user files.

UserFileReadBytes (4 bytes): A 32-bit unsigned integer value containing the number of bytes read from user files.

UserDiskReads (4 bytes): A 32-bit unsigned integer value containing the number of read operations on user files that went to the disk rather than the cache. This value includes **sub-read** operations.

UserFileWrites (4 bytes): A 32-bit unsigned integer value containing the number of write operations on user files.

UserFileWriteBytes (4 bytes): A 32-bit unsigned integer value containing the number of bytes written to user files.

UserDiskWrites (4 bytes): A 32-bit unsigned integer value containing the number of write operations on user files that went to disk rather than the cache. This value includes sub-write operations.

MetaDataReads (4 bytes): A 32-bit unsigned integer value containing the number of read operations on metadata files.

MetaDataReadBytes (4 bytes): A 32-bit unsigned integer value containing the number of bytes read from metadata files.

MetaDataDiskReads (4 bytes): A 32-bit unsigned integer value containing the number of read operations on metadata files. This value includes sub-read operations.

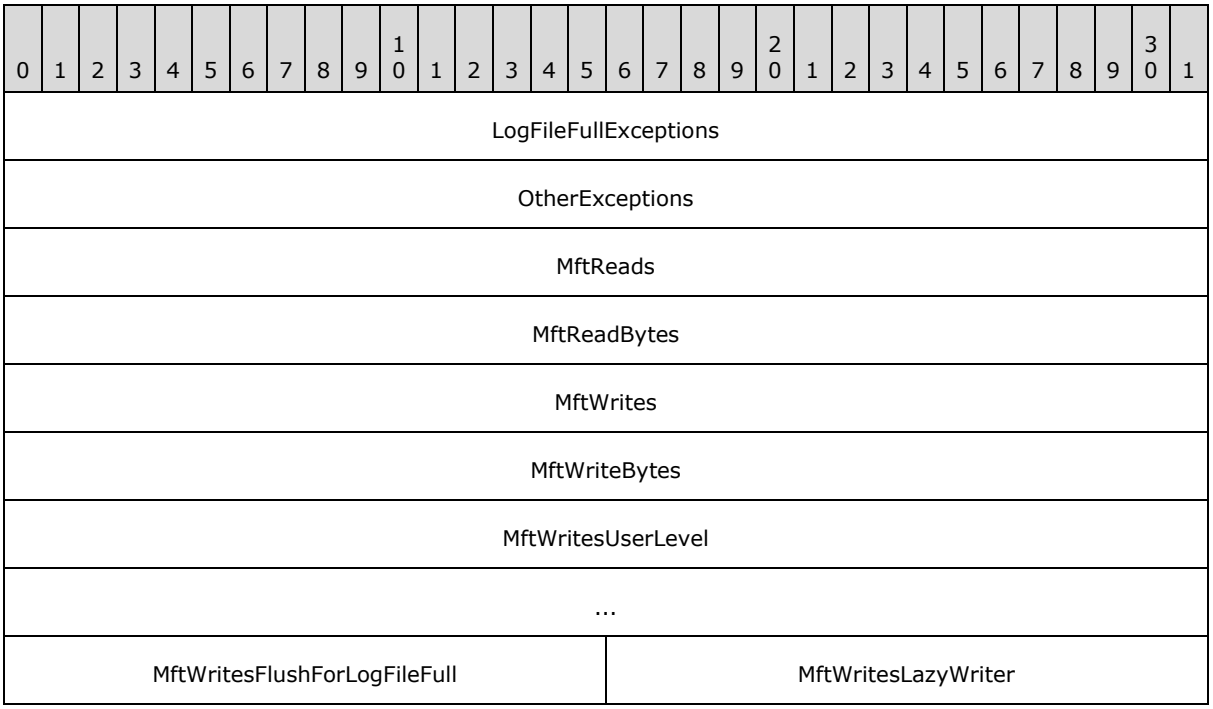
MetaDataWrites (4 bytes): A 32-bit unsigned integer value containing the number of write operations on metadata files.

MetaDataWriteBytes (4 bytes): A 32-bit unsigned integer value containing the number of bytes written to metadata files.

MetaDataDiskWrites (4 bytes): A 32-bit unsigned integer value containing the number of write operations on metadata files. This value includes sub-write operations.

2.3.8.2 NTFS_STATISTICS

The NTFS_STATISTICS data element is returned with a FSCTL_FILESYSTEM_GET_STATISTICS reply message when NTFS file system statistics are requested. The NTFS_STATISTICS data element is as follows:



MftWritesUserRequest	Padding1
Mft2Writes	
Mft2WriteBytes	
Mft2WritesUserLevel	
...	
Mft2WritesFlushForLogFileFull	Mft2WritesLazyWriter
Mft2WritesUserRequest	Padding2
RootIndexReads	
RootIndexReadBytes	
RootIndexWrites	
RootIndexWriteBytes	
BitmapReads	
BitmapReadBytes	
BitmapWrites	
BitmapWriteBytes	
BitmapWritesFlushForLogFileFull	BitmapWritesLazyWriter
BitmapWritesUserRequest	BitmapWritesUserLevel
...	
MftBitmapReads	
MftBitmapReadBytes	
MftBitmapWrites	
MftBitmapWriteBytes	
MftBitmapWritesFlushForLogFileFull	MftBitmapWritesLazyWriter

MftBitmapWritesUserRequest	MftBitmapWritesUserLevel
...	
...	Padding3
UserIndexReads	
UserIndexReadBytes	
UserIndexWrites	
UserIndexWriteBytes	
LogFileReads	
LogFileReadBytes	
LogFileWrites	
LogFileWriteBytes	
Allocate	
...	
...	
...	
...	
...	
...	
...	
(Allocate cont'd for 2 rows)	

LogFileFullExceptions (4 bytes): A 32-bit unsigned integer value containing the number of exceptions generated due to the log file being full.

OtherExceptions (4 bytes): A 32-bit unsigned integer value containing the number of other exceptions generated.

MftReads (4 bytes): A 32-bit unsigned integer value containing the number of read operations on the **Master File Table (MFT)**.

MftReadBytes (4 bytes): A 32-bit unsigned integer value containing the number of bytes read from the MFT.

MftWrites (4 bytes): A 32-bit unsigned integer value containing the number of write operations on the MFT.

MftWriteBytes (4 bytes): A 32-bit unsigned integer value containing the number of bytes written to the MFT.

MftWritesUserLevel (8 bytes): An MftWritesUserLevel structure containing statistics about writes resulting from certain user-level operations.

MftWritesFlushForLogFileFull (2 bytes): A 16-bit unsigned integer containing the number of flushes of the MFT performed because the log file was full.

MftWritesLazyWriter (2 bytes): A 16-bit unsigned integer containing the number of MFT write operations performed by the lazy writer thread.

MftWritesUserRequest (2 bytes): A 16-bit unsigned integer that is the sum of the four fields in the MftWritesUserLevel structure.

Padding1 (2 bytes): Unused. This field SHOULD be set to 0 and MUST be ignored.

Mft2Writes (4 bytes): A 32-bit unsigned integer value containing the number of write operations on the **master file table mirror (MFT2)**.

Mft2WriteBytes (4 bytes): A 32-bit unsigned integer value containing the number of bytes written to the MFT2.

Mft2WritesUserLevel (8 bytes): An MftWritesUserLevel structure containing statistics about writes resulting from certain user-level operations.

Mft2WritesFlushForLogFileFull (2 bytes): A 16-bit unsigned integer containing the number of flushes of the MFT2 performed because the log file was full.

Mft2WritesLazyWriter (2 bytes): A 16-bit unsigned integer containing the number of MFT2 write operations performed by the lazy writer thread.

Mft2WritesUserRequest (2 bytes): A 16-bit unsigned integer that contains the sum of the four fields in the Mft2WritesUserLevel structure.

Padding2 (2 bytes): Unused. This field SHOULD be set to 0 and MUST be ignored.

RootIndexReads (4 bytes): A 32-bit unsigned integer value containing the number of read operations on the root index.

RootIndexReadBytes (4 bytes): A 32-bit unsigned integer value containing the number of bytes read from the root index.

RootIndexWrites (4 bytes): A 32-bit unsigned integer value containing the number of write operations on the root index.

RootIndexWriteBytes (4 bytes): A 32-bit unsigned integer value containing the number of bytes written to the root index.

BitmapReads (4 bytes): A 32-bit unsigned integer value containing the number of read operations on the cluster allocation bitmap.

BitmapReadBytes (4 bytes): A 32-bit unsigned integer value containing the number of bytes read from the cluster allocation bitmap.

BitmapWrites (4 bytes): A 32-bit unsigned integer value containing the number of write operations on the cluster allocation bitmap. This is the sum of the **BitmapWritesFlushForLogFileFull**, **BitmapWritesLazyWriter** and **BitmapWritesUserRequest** fields.

BitmapWriteBytes (4 bytes): A 32-bit unsigned integer value containing the number of bytes written to the cluster allocation bitmap.

BitmapWritesFlushForLogFileFull (2 bytes): A 16-bit unsigned integer containing the number of flushes of the bitmap performed because the log file was full.

BitmapWritesLazyWriter (2 bytes): A 16-bit unsigned integer containing the number of bitmap write operations performed by the lazy writer thread.

BitmapWritesUserRequest (2 bytes): A 16-bit unsigned integer that is the sum of the fields in the BitmapWritesUserLevel structure.

BitmapWritesUserLevel (6 bytes): A BitmapWritesUserLevel structure containing statistics about bitmap writes resulting from certain user-level operations.

MftBitmapReads (4 bytes): A 32-bit unsigned integer value containing the number of read operations on the MFT bitmap.

MftBitmapReadBytes (4 bytes): A 32-bit unsigned integer value containing the number of bytes read from the MFT bitmap.

MftBitmapWrites (4 bytes): A 32-bit unsigned integer value containing the number of write operations on the MFT bitmap. This value is the sum of the **MftBitmapWritesFlushForLogFileFull**, **MftBitmapWritesLazyWriter** and **MftBitmapWritesUserRequest** fields.

MftBitmapWriteBytes (4 bytes): A 32-bit unsigned integer value containing the number of bytes written to the MFT bitmap.

MftBitmapWritesFlushForLogFileFull (2 bytes): A 16-bit unsigned integer containing the number of flushes of the MFT bitmap performed because the log file was full.

MftBitmapWritesLazyWriter (2 bytes): A 16-bit unsigned integer value containing the number of MFT bitmap write operations performed by the lazy writer thread.

MftBitmapWritesUserRequest (2 bytes): A 16-bit unsigned integer that is the sum of all the fields in the MftBitmapWritesUserLevel structure.

MftBitmapWritesUserLevel (8 bytes): An MftBitmapWritesUserLevel structure containing statistics about MFT bitmap writes resulting from certain user-level operations.

Padding3 (2 bytes): Unused. This field SHOULD be set to 0 and MUST be ignored.

UserIndexReads (4 bytes): A 32-bit unsigned integer value containing the number of read operations on the user index.

UserIndexReadBytes (4 bytes): A 32-bit unsigned integer value containing the number of bytes read from user indices.

UserIndexWrites (4 bytes): A 32-bit unsigned integer value containing the number of write operations on user indices.

UserIndexWriteBytes (4 bytes): A 32-bit unsigned integer value containing the number of bytes written to user indices.

LogFileReads (4 bytes): A 32-bit unsigned integer value containing the number of read operations on the log file.

LogFileReadBytes (4 bytes): A 32-bit unsigned integer value containing the number of bytes read from the log file.

LogFileWrites (4 bytes): A 32-bit unsigned integer value containing the number of write operations on the log file.

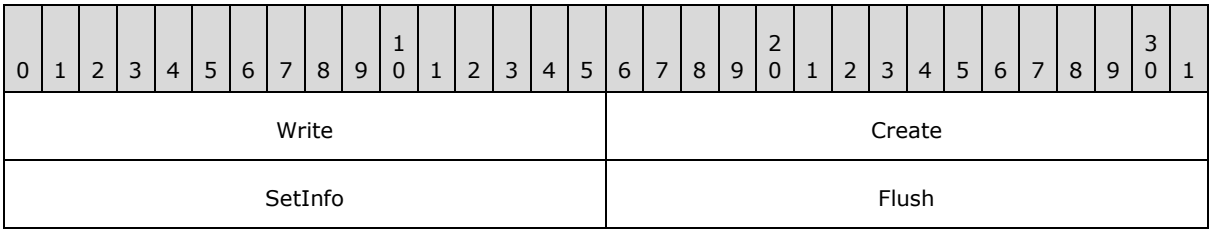
LogFileWriteBytes (4 bytes): A 32-bit unsigned integer value containing the number of bytes written to the log file.

Allocate (40 bytes): An Allocate structure describes cluster allocation patterns in NTFS.

2.3.8.2.1 MftWritesUserLevel

The MftWritesUserLevel structure contains statistics about writes resulting from certain user-level operations.

The MftWritesUserLevel structure is as follows.



Write (2 bytes): A 16-bit unsigned integer containing the number of MFT writes due to a write operation.

Create (2 bytes): A 16-bit unsigned integer containing the number of MFT writes due to a create operation.

SetInfo (2 bytes): A 16-bit unsigned integer containing the number of MFT writes due to a set file information operation.

Flush (2 bytes): A 16-bit unsigned integer containing the number of MFT writes due to a flush operation.

2.3.8.2.2 Mft2WritesUserLevel

The Mft2WritesUserLevel structure contains statistics about writes resulting from certain user-level operations.

The Mft2WritesUserLevel structure is as follows.

0	1	2	3	4	5	6	7	8	9	10	1	2	3	4	5	6	7	8	9	20	1	2	3	4	5	6	7	8	9	30	1
Write																Create															
SetInfo																Flush															

Write (2 bytes): A 16-bit unsigned integer containing the number of MFT2 writes due to a write operation.

Create (2 bytes): A 16-bit unsigned integer containing the number of MFT2 writes due to a create operation.

SetInfo (2 bytes): A16-bit unsigned integer containing the number of MFT2 writes due to a set file information operation.

Flush (2 bytes): A 16-bit unsigned integer containing the number of MFT2 writes due to a flush operation.

2.3.8.2.3 BitmapWritesUserLevel

The BitmapWritesUserLevel structure contains statistics about bitmap writes resulting from certain user-level operations.

The BitmapWritesUserLevel structure is as follows.

0	1	2	3	4	5	6	7	8	9	10	1	2	3	4	5	6	7	8	9	20	1	2	3	4	5	6	7	8	9	30	1
Write																Create															
SetInfo																															

Write (2 bytes): A 16-bit unsigned integer containing the number of bitmap writes due to a write operation.

Create (2 bytes): A 16-bit unsigned integer containing the number of bitmap writes due to a create operation.

SetInfo (2 bytes): A 16-bit unsigned integer containing the number of bitmap writes due to a set file information operation.

2.3.8.2.4 MftBitmapWritesUserLevel

The MftBitmapWritesUserLevel structure contains statistics about MFT bitmap write operations resulting from certain user-level operations.

The MftBitmapWritesUserLevel structure is as follows.

0	1	2	3	4	5	6	7	8	9	10	1	2	3	4	5	6	7	8	9	20	1	2	3	4	5	6	7	8	9	30	1
Write																Create															
SetInfo																Flush															

Write (2 bytes): A 16-bit unsigned integer containing the number of MFT bitmap write operations due to a write operation.

Create (2 bytes): A 16-bit unsigned integer containing the number of MFT bitmap write operations due to a create operation.

SetInfo (2 bytes): A 16-bit unsigned integer containing the number of MFT bitmap write operations due to a set file information operation.

Flush (2 bytes): A 16-bit unsigned integer containing the number of MFT bitmap write operations due to a flush operation.

2.3.8.2.5 Allocate

The Allocate structure describes cluster allocation patterns in NTFS. The cache refers to in-memory structures that allow quick lookups of free cluster runs either by **logical cluster number (LCN)** or by run length.

The Allocate structure is as follows.

0	1	2	3	4	5	6	7	8	9	10	1	2	3	4	5	6	7	8	9	20	1	2	3	4	5	6	7	8	9	30	1
Calls																															
Clusters																															
Hints																															
RunsReturned																															
HintsHonored																															
HintsClusters																															
Cache																															
CacheClusters																															
CacheMiss																															
CacheMissClusters																															

Calls (4 bytes): A 32-bit unsigned integer value containing the number of individual calls to allocate clusters.

Clusters (4 bytes): A 32-bit unsigned integer value containing the number of clusters allocated.

Hints (4 bytes): A 32-bit unsigned integer value containing the number of times a hint was specified when trying to determine which clusters to allocate.

RunsReturned (4 bytes): A 32-bit unsigned integer value containing the number of runs used to satisfy all the requests.

HintsHonored (4 bytes): A 32-bit unsigned integer value containing the number of times the starting LCN hint was used to determine which clusters to allocate.

HintsClusters (4 bytes): A 32-bit unsigned integer value containing the number of clusters allocated via the starting LCN hint.

Cache (4 bytes): A 32-bit unsigned integer value containing the number of times the run length cache was useful.

CacheClusters (4 bytes): A 32-bit unsigned integer value containing the number of clusters allocated via the run length cache.

CacheMiss (4 bytes): A 32-bit unsigned integer value containing the number of times the cache was not useful and the bitmapped had to be scanned for free clusters.

CacheMissClusters (4 bytes): A 32-bit unsigned integer value containing the number of clusters allocated by scanning the bitmap.

2.3.8.3 FAT_STATISTICS

The FAT_STATISTICS data element is returned with a FSCTL_FILESYSTEM_GET_STATISTICS reply message when FAT file system statistics are requested. The FAT_STATISTICS data element is as follows:

0	1	2	3	4	5	6	7	8	9	10	1	2	3	4	5	6	7	8	9	20	1	2	3	4	5	6	7	8	9	30	1
CreateHits																															
SuccessfulCreates																															
FailedCreates																															
NonCachedReads																															
NonCachedReadBytes																															
NonCachedWrites																															
NonCachedWriteBytes																															

NonCachedDiskReads
NonCachedDiskWrites

CreateHits (4 bytes): A 32-bit unsigned integer value containing the number of create operations.

SuccessfulCreates (4 bytes): A 32-bit unsigned integer value containing the number of successful create operations.

FailedCreates (4 bytes): A 32-bit unsigned integer value containing the number of failed create operations.

NonCachedReads (4 bytes): A 32-bit unsigned integer value containing the number of read operations that were not cached.

NonCachedReadBytes (4 bytes): A 32-bit unsigned integer value containing the number of bytes read from a file that were not cached.

NonCachedWrites (4 bytes): A 32-bit unsigned integer value containing the number of write operations that were not cached.

NonCachedWriteBytes (4 bytes): A 32-bit unsigned integer value containing the number of bytes written to a file that were not cached.

NonCachedDiskReads (4 bytes): A 32-bit unsigned integer value containing the number of read operations that were not cached. This value includes sub-read operations.

NonCachedDiskWrites (4 bytes): A 32-bit unsigned integer value containing the number of write operations that were not cached. This value includes sub-write operations.

2.3.8.4 EXFAT_STATISTICS

The EXFAT_STATISTICS data element is returned with a FSCTL_FILESYSTEM_GET_STATISTICS reply message when exFAT file system statistics are requested. The EXFAT_STATISTICS data element is as follows:

0	1	2	3	4	5	6	7	8	9	10	1	2	3	4	5	6	7	8	9	20	1	2	3	4	5	6	7	8	9	30	1
CreateHits																															
SuccessfulCreates																															
FailedCreates																															
NonCachedReads																															
NonCachedReadBytes																															
NonCachedWrites																															

NonCachedWriteBytes
NonCachedDiskReads
NonCachedDiskWrites

CreateHits (4 bytes): A 32-bit unsigned integer value containing the number of create operations.

SuccessfulCreates (4 bytes): A 32-bit unsigned integer value containing the number of successful create operations.

FailedCreates (4 bytes): A 32-bit unsigned integer value containing the number of failed create operations.

NonCachedReads (4 bytes): A 32-bit unsigned integer value containing the number of read operations that were not cached.

NonCachedReadBytes (4 bytes): A 32-bit unsigned integer value containing the number of bytes read from a file that were not cached.

NonCachedWrites (4 bytes): A 32-bit unsigned integer value containing the number of write operations that were not cached.

NonCachedWriteBytes (4 bytes): A 32-bit unsigned integer value containing the number of bytes written to a file that were not cached.

NonCachedDiskReads (4 bytes): A 32-bit unsigned integer value containing the number of read operations that were not cached. This value includes sub-read operations.

NonCachedDiskWrites (4 bytes): A 32-bit unsigned integer value containing the number of write operations that were not cached. This value includes sub-write operations.

2.3.9 FSCTL_FIND_FILES_BY_SID Request

The FSCTL_FIND_FILES_BY_SID Request message requests that the server return a list of the files and directories whose owner matches the specified **security identifier (SID)**, in no necessary order. The search spans the file system subtree descending from the directory associated with the handle on which this FSCTL was invoked. This message contains a FIND_BY_SID_DATA data element.

The FIND_BY_SID_DATA data element is as follows.

0	1	2	3	4	5	6	7	8	9	10	1	2	3	4	5	6	7	8	9	20	1	2	3	4	5	6	7	8	9	30	1
Restart																															
SID (variable)																															
...																															

Restart (4 bytes): A 32-bit unsigned integer value that indicates to restart the search. This value MUST be 0x00000001 on the first call so that the search starts from the beginning of the directory on which the operation is requested. For subsequent calls, this member SHOULD be zero so that the search resumes at the point where it stopped.

SID (variable): A SID ([MS-DTYP] section 2.4.2.2) data element that specifies the owner.

2.3.10 FSCTL_FIND_FILES_BY_SID Reply

The FSCTL_FIND_FILES_BY_SID Reply message returns the results of the FSCTL_FIND_FILES_BY_SID Request (section 2.3.9) as an array of FILE_NAME_INFORMATION (section 2.1.7) data elements, one for each matching file or directory that is found, in no necessary order. All returned file names MUST be relative to the directory on which the FSCTL_FIND_FILES_BY_SID Request was issued. This returns as many FILE_NAME_INFORMATION data elements as will fit in the provided output buffer. The beginning of each FILE_NAME_INFORMATION data element MUST be aligned to an 8-byte boundary, as measured from the beginning of the buffer. The last FILE_NAME_INFORMATION structure returned MAY<16> contain trailing padding.

This message also returns a status code, as specified in [MS-ERREF] section 2.3. The status code returned directly by the function that processes this FSCTL MUST be STATUS_SUCCESS or one of the following.

Status code	Meaning
STATUS_NO_QUOTAS_FOR_ACCOUNT 0x0000010D	Quota tracking is not enabled; therefore, the file system does not keep a record of file owners. This is considered a success code. The reply MUST NOT contain any data elements.
STATUS_INVALID_PARAMETER 0xC000000D	The handle specified is not the handle to a directory.
STATUS_ACCESS_DENIED 0xC0000022	Neither the SeManageVolumePrivilege nor the SeBackupPrivilege privilege is held.
STATUS_BUFFER_TOO_SMALL 0xC0000023	The output buffer is not large enough to contain the FILE_NAME_INFORMATION structure (including any trailing padding) for the first matching file or directory.
STATUS_INVALID_USER_BUFFER 0xC00000E8	The input buffer is less than the size of a long integer (4 bytes) plus the length of the SID provided, or the input or output buffer is not aligned to the native word size of the platform, or the size of the output buffer is less than the minimum size of a FILE_NAME_INFORMATION structure (8 bytes), or the restart value is greater than 1.

When the status code is STATUS_SUCCESS, the responder MUST retain an implementation-dependent indication of where the directory processing ended, which is required to support a subsequent FSCTL_FIND_FILES_BY_SID Request with the **Restart** field set to 0x00000000. For an example of FSCTL_FIND_FILES_BY_SID restart handling, see the [MS-FSA] section titled "FSCTL_FIND_FILES_BY_SID".

2.3.11 FSCTL_GET_COMPRESSION Request

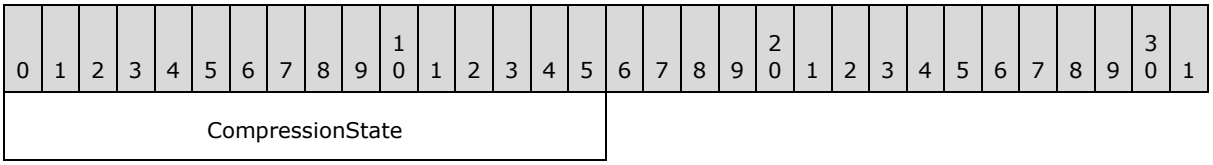
This message requests that the server return the current compression state of the file or directory associated with the handle on which this FSCTL was invoked.

This message does not contain any additional data elements.

2.3.12 FSCTL_GET_COMPRESSION Reply

The FSCTL_GET_COMPRESSION reply message returns the results of the FSCTL_GET_COMPRESSION request as a 16-bit unsigned integer value that indicates the current compression state of the file or directory.

The CompressionState element is as follows.



CompressionState (2 bytes): One of the following standard values MUST be returned.

Value	Meaning
COMPRESSION_FORMAT_NONE 0x0000	The file or directory is not compressed.
COMPRESSION_FORMAT_LZNT1 0x0002	The file or directory is compressed by using the LZNT1 compression algorithm. For more information, see [UASDC].
All other values	Reserved for future use and MUST NOT be used.

The actual file or directory compression format is implementation-dependent.<17>

If the file system of the volume that contains the specified file or directory does not support per-file or per-directory compression, the request MUST NOT succeed. The error code that is returned in this situation MUST be as specified in section 2.2.

This message also returns a status code, as specified in [MS-ERREF] section 2.3. The status code that is returned directly by the function that processes this FSCTL MUST be STATUS_SUCCESS or one of the following.

Error code	Meaning
STATUS_INVALID_PARAMETER 0xC000000D	The output buffer length is less than 2, or the handle is not to a file or directory.
STATUS_INVALID_DEVICE_REQUEST 0xC0000010	The volume does not support compression.<18>

2.3.13 FSCTL_GET_NTFS_VOLUME_DATA Request

This message requests that the server return information about the NTFS file system volume that contains the file or directory that is associated with the handle on which this FSCTL was invoked.

This message does not contain any parameters.

2.3.14 FSCTL_GET_NTFS_VOLUME_DATA Reply

The FSCTL_GET_NTFS_VOLUME_DATA reply message returns the results of the FSCTL_GET_NTFS_VOLUME_DATA request as an NTFS_VOLUME_DATA_BUFFER element.

The NTFS_VOLUME_DATA_BUFFER contains information on a volume. For more information about the NTFS file system, see [MSFT-NTFS].

0	1	2	3	4	5	6	7	8	9	10	1	2	3	4	5	6	7	8	9	20	1	2	3	4	5	6	7	8	9	30	1
VolumeSerialNumber																															
...																															
NumberSectors																															
...																															
TotalClusters																															
...																															
FreeClusters																															
...																															
TotalReserved																															
...																															
BytesPerSector																															
BytesPerCluster																															
BytesPerFileRecordSegment																															
ClustersPerFileRecordSegment																															
MftValidDataLength																															
...																															
MftStartLcn																															
...																															
Mft2StartLcn																															

...
MftZoneStart
...
MftZoneEnd
...

VolumeSerialNumber (8 bytes): A 64-bit signed integer that contains the serial number of the volume. This is a unique number assigned to the volume media by the operating system when the volume is formatted.

NumberSectors (8 bytes): A 64-bit signed integer that contains the number of **sectors** in the specified volume.

TotalClusters (8 bytes): A 64-bit signed integer that contains the total number of **clusters** in the specified volume.

FreeClusters (8 bytes): A 64-bit signed integer that contains the number of free clusters in the specified volume.

TotalReserved (8 bytes): A 64-bit signed integer that contains the number of reserved clusters in the specified volume. Reserved clusters are free clusters reserved for when the volume becomes full. Reserved clusters are released when either the master file table grows beyond its allocated space (the volume has a large number of small files) or the volume becomes full (the volume has a small number of large files).

BytesPerSector (4 bytes): A 32-bit unsigned integer that contains the number of bytes in a sector on the specified volume.

BytesPerCluster (4 bytes): A 32-bit unsigned integer that contains the number of bytes in a cluster on the specified volume. This value is also known as the cluster factor.

BytesPerFileRecordSegment (4 bytes): A 32-bit unsigned integer that contains the number of bytes in a **file record segment**.

ClustersPerFileRecordSegment (4 bytes): A 32-bit unsigned integer that contains the number of clusters in a file record segment.

MftValidDataLength (8 bytes): A 64-bit signed integer that contains the size of the master file table in bytes.

MftStartLcn (8 bytes): A 64-bit signed integer that contains the starting logical cluster number (LCN) of the master file table.

Mft2StartLcn (8 bytes): A 64-bit signed integer that contains the starting logical cluster number of the master file table mirror.

MftZoneStart (8 bytes): A 64-bit signed integer that contains the starting logical cluster number of the master file table zone.

MftZoneEnd (8 bytes): A 64-bit signed integer that contains the ending logical cluster number of the master file table zone. The size of the master file table zone is (**MftZoneEnd** - **MftZoneStart**) clusters.

This message also returns a status code, as specified in [MS-ERREF] section 2.3. The status code returned directly by the function that processes this FSCTL MUST be STATUS_SUCCESS or one of the following.

Error code	Meaning
STATUS_INVALID_PARAMETER 0xC000000D	The handle specified is not open.
STATUS_VOLUME_DISMOUNTED 0xC000026E	The specified volume is no longer mounted.
STATUS_BUFFER_TOO_SMALL 0xC0000023	The output buffer is too small to contain an **NTFS_VOLUME_DATA_BUFFER** structure.

2.3.15 FSCTL_GET_OBJECT_ID Request

This message requests that the server return the object identifier for the file or directory associated with the handle on which this FSCTL was invoked.

Object identifiers are 16-byte opaque values that are used to track files and directories, and they are generated by the server. File and directory object identifiers are invisible to most applications and should never be modified by applications.

This message does not contain any additional data elements.

2.3.16 FSCTL_GET_OBJECT_ID Reply

This message returns the results of an FSCTL_GET_OBJECT_ID request in a FILE_OBJECTID_BUFFER (section 2.1.3).

If the file system of the volume containing the specified file or directory does not support the use of object IDs, the request will not succeed. The error code returned in this situation is specified in section 2.2.

This message also returns a status code, as specified in [MS-ERREF] section 2.3. The status code returned directly by the function that processes this FSCTL MUST be STATUS_SUCCESS or one of the following.

Error code	Meaning
STATUS_INVALID_PARAMETER 0xC000000D	The output buffer length is less than the size of a FILE_OBJECTID_BUFFER or the handle is not to a file or directory.
STATUS_OBJECTID_NOT_FOUND 0xC00002F0	The file or directory has no object ID.
STATUS_INVALID_DEVICE_REQUEST 0xC0000010	The file system does not support the use of object IDs.

2.3.17 FSCTL_GET_REPARSE_POINT Request

This message requests that the server return the reparse point data for the file or directory associated with the handle on which this FSCTL was invoked.

This message MUST only be sent for a file or directory handle.

This message does not contain any additional data elements.

2.3.18 FSCTL_GET_REPARSE_POINT Reply

This message returns the results of the FSCTL_GET_REPARSE_POINT request. The message contains a REPARSE_GUID_DATA_BUFFER (including subtypes) or a REPARSE_DATA_BUFFER data element.

Both the REPARSE_GUID_DATA_BUFFER and the REPARSE_DATA_BUFFER structures begin with a **ReparseTag** field. The ReparseTag value uniquely identifies the filter driver that creates/uses the reparse point, and the application's filter driver processes the reparse point data as either a REPARSE_GUID_DATA_BUFFER or a REPARSE_DATA_BUFFER, depending on the structure implemented by the filter driver for that type of reparse point. A particular filter driver is implemented with specific support for the type of reparse point data structure it accepts.

If the file system of the volume containing the specified file or directory does not support the use of reparse points, the request will not succeed. The error code returned in this situation MAY vary, depending on the file system.<19>

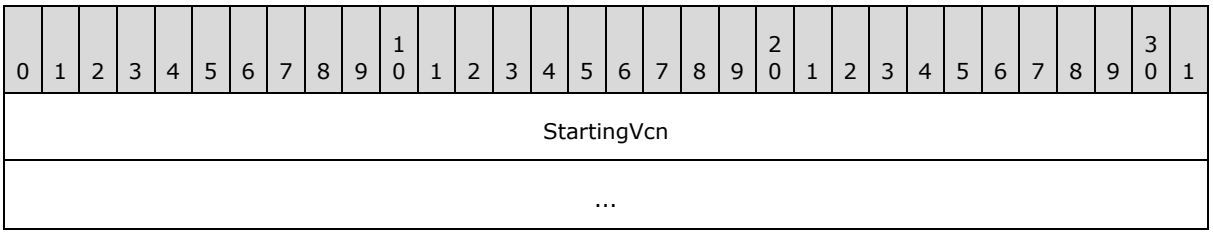
This message returns a status code, as specified in [MS-ERREF] section 2.3. The status code returned directly by the function that processes this FSCTL MUST be STATUS_SUCCESS or one of the following.

Error code	Meaning
STATUS_BUFFER_TOO_SMALL 0xC0000023	The output buffer is too small to contain a REPARSE_GUID_DATA_BUFFER.
STATUS_INVALID_PARAMETER 0xC000000D	The handle is not to a file or directory.
STATUS_BUFFER_OVERFLOW 0x80000005	The output buffer filled before all the reparse point data was returned.
STATUS_NOT_A_REPARSE_POINT 0xC0000275	The file or directory is not a reparse point.
STATUS_INVALID_DEVICE_REQUEST 0xC0000010	The file system does not support the use of reparse points.

2.3.19 FSCTL_GET_RETRIEVAL_POINTERS Request

The FSCTL_GET_RETRIEVAL_POINTERS request message requests that the server return a list of extents for the file or directory associated with the handle on which this FSCTL was invoked. The extents describe the mapping between **virtual cluster numbers (VCNs)** and logical cluster numbers (LCNs). This request is most commonly used by defragmentation utilities. This message contains a STARTING_VCN_INPUT_BUFFER data element.

The STARTING_VCN_INPUT_BUFFER data element is as follows.



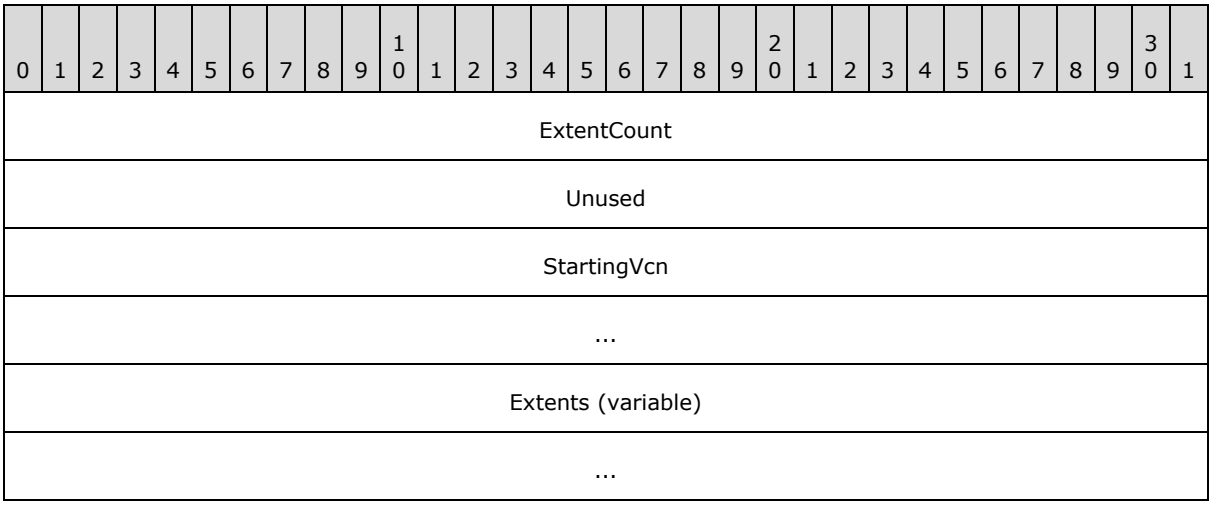
StartingVcn (8 bytes): A 64-bit signed integer that contains the virtual cluster number (VCN) at which to begin retrieving extents in the file. This value MUST be greater than or equal to 0.

2.3.20 FSCTL_GET_RETRIEVAL_POINTERS Reply

The FSCTL_GET_RETRIEVAL_POINTERS reply message returns the results of the FSCTL_GET_RETRIEVAL_POINTERS request as a variably–sized data element, RETRIEVAL_POINTERS_BUFFER, that specifies the allocation and location on disk of a specific file.

The FSCTL_GET_RETRIEVAL_POINTERS reply returns the extent locations (that is, locations of allocated regions of disk space) of nonresident data. A file system MAY allow resident data, which is data that can be written to disk within the file's directory record. Because resident data requires no additional disk space allocation, no extent locations are associated with resident data.<20>

The RETRIEVAL_POINTERS_BUFFER data element is as follows.



ExtentCount (4 bytes): A 32-bit unsigned integer that contains the number of EXTENTS data elements in the **Extents** array. This number can be zero if there are no clusters allocated at (or beyond) the specified **StartingVcn**.

Unused (4 bytes): Reserved for alignment. This field can contain any value and MUST be ignored.

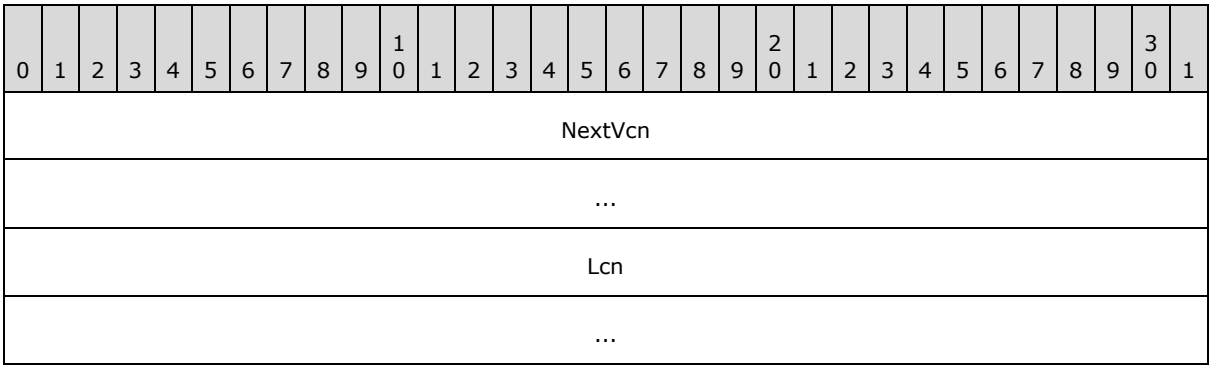
StartingVcn (8 bytes): A 64-bit signed integer that contains the starting virtual cluster number (VCN) returned by the FSCTL_GET_RETRIEVAL_POINTERS reply. This is not necessarily the VCN requested by the FSCTL_GET_RETRIEVAL_POINTERS request, as the file system driver

might return the starting VCN of the extent containing the requested starting VCN. This value MUST be greater than or equal to 0.

Extents (variable): An array of zero or more EXTENTS data elements. For the number of EXTENTS data elements in the array, see **ExtentCount**.

2.3.20.1 EXTENTS

The EXTENTS data element is as follows.



NextVcn (8 bytes): A 64-bit signed integer that contains the VCN at which the next extent begins. This value minus either **StartingVcn** (for the first **Extents** array element) or the **NextVcn** of the previous element of the array (for all other **Extents** array elements) is the length in clusters of the current extent.

Lcn (8 bytes): A 64-bit signed integer that contains the logical cluster number (LCN) at which the current extent begins on the volume. A 64-bit value of -1 indicates either a **compression unit** that is partially allocated or an unallocated region of a **sparse file**. For more information about sparse files, see [SPARSE]. Compression is performed in 16-cluster units. If a given 16-cluster unit compresses to fit in, for example, 9 clusters, there will be a 7-cluster extent of the file with an LCN of -1.

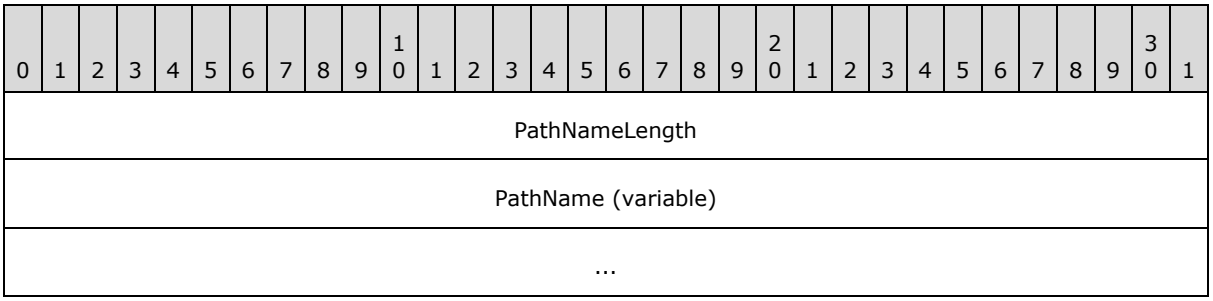
This message returns a status code, as specified in [MS-ERREF] section 2.3. The status code returned directly by the function that processes this FSCTL MUST be STATUS_SUCCESS or one of the following.

Error code	Meaning
STATUS_BUFFER_TOO_SMALL 0xC0000023	The output buffer is too small to contain a RETRIEVAL_POINTERS_BUFFER structure.
STATUS_INVALID_PARAMETER 0xC000000D	The input buffer is too small to contain a STARTING_VCN_INPUT_BUFFER, or the **StartingVcn** given is negative, or the handle is not to a file or directory.
STATUS_END_OF_FILE 0xC0000011	The stream is resident in the MFT and has no clusters allocated, or the starting VCN is beyond the end of the file.
STATUS_BUFFER_OVERFLOW 0x80000005	The output buffer filled before all the extents for this file were returned.

2.3.21 FSCTL_IS_PATHNAME_VALID Request

The FSCTL_IS_PATHNAME_VALID request message requests that the server indicate whether the specified pathname is well-formed (of acceptable length, with no invalid characters, and so on - see section 2.1.5) with respect to the volume that contains the file or directory associated with the handle on which this FSCTL was invoked.

The data element is as follows.



PathNameLength (4 bytes): An unsigned 32-bit integer that specifies the length, in bytes, of the **PathName** data element.

PathName (variable): A variable-length Unicode string that specifies the path name.

2.3.22 FSCTL_IS_PATHNAME_VALID Reply

This message returns the results of the FSCTL_IS_PATHNAME_VALID Request (section 2.3.21).

A STATUS_SUCCESS from this call means that the pathname is valid. An error means that the pathname is not valid.<21>

2.3.23 FSCTL_LMR_SET_LINK_TRACKING_INFORMATION Request

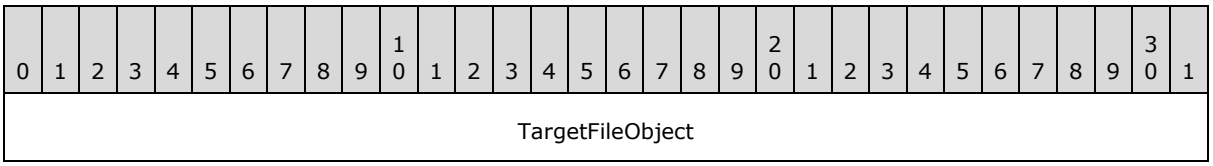
The FSCTL_LMR_SET_LINK_TRACKING_INFORMATION request message sets **Distributed Link Tracking** information such as file system type, volume ID, object ID, and destination computer's **NetBIOS name** for the file or directory associated with the handle on which this FSCTL was invoked. For more information about Distributed Link Tracking, see [MS-DLTW] section 3.1.6.

There are two variations of this request, depending on whether it is embedded within [MS-SMB] or [MS-SMB2]. The request definitions are as follows.

- FSCTL_LMR_SET_LINK_TRACKING_INFORMATION Request for SMB
- FSCTL_LMR_SET_LINK_TRACKING_INFORMATION Request for SMB2

2.3.23.1 FSCTL_LMR_SET_LINK_TRACKING_INFORMATION Request for SMB

The message contains a REMOTE_LINK_TRACKING_INFORMATION32 data element. The SMB REMOTE_LINK_TRACKING_INFORMATION32 data element is as follows.



TargetLinkTrackingInformationLength
TargetLinkTrackingInformationBuffer (variable)
...

TargetFileObject (4 bytes): The **Fid** of the file from which to obtain link tracking information. For Fid type, see [MS-SMB] section 2.2.7.2.1.

TargetLinkTrackingInformationLength (4 bytes): The length of the **TargetLinkTrackingInformationBuffer**.

TargetLinkTrackingInformationBuffer (variable): This field is as specified in TARGET_LINK_TRACKING_INFORMATION_Buffer.

2.3.23.2 FSCTL_LMR_SET_LINK_TRACKING_INFORMATION Request for SMB2

The message contains an SMB2_REMOTE_LINK_TRACKING_INFORMATION data element. The SMB2_REMOTE_LINK_TRACKING_INFORMATION data element is as follows.

0	1	2	3	4	5	6	7	8	9	10	1	2	3	4	5	6	7	8	9	20	1	2	3	4	5	6	7	8	9	30	1
TargetFileObject																															
...																															
TargetLinkTrackingInformationLength																															
TargetLinkTrackingInformationBuffer (variable)																															
...																															

TargetFileObject (8 bytes): Nonzero values of **TargetFileObject** are never used in the Server Message Block (SMB) Version 2 Protocol variant of the request. This field MUST be set to zero.

TargetLinkTrackingInformationLength (4 bytes): The length of the **TargetLinkTrackingInformationBuffer** field.

TargetLinkTrackingInformationBuffer (variable): This field is as specified in TARGET_LINK_TRACKING_INFORMATION_BUFFER.

2.3.23.3 TARGET_LINK_TRACKING_INFORMATION_Buffer

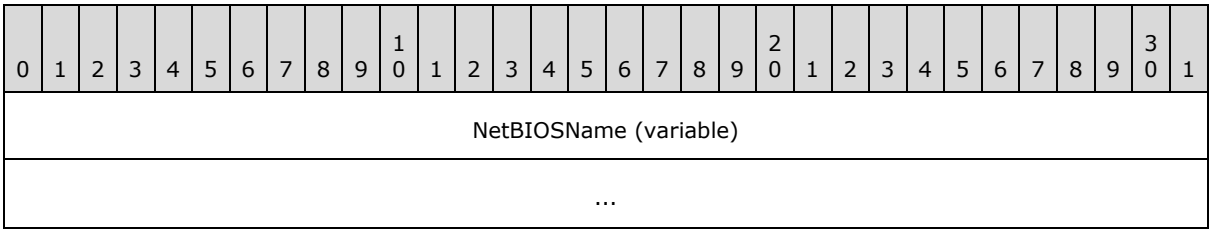
The TARGET_LINK_TRACKING_INFORMATION_Buffer data element MUST take one of the following forms:

- TARGET_LINK_TRACKING_INFORMATION_Buffer_1 if the **TargetLinkTrackingInformationLength** value is less than 36.

- TARGET_LINK_TRACKING_INFORMATION_Buffer_2 if the **TargetLinkTrackingInformationLength** value is greater than or equal to 36.

2.3.23.3.1 TARGET_LINK_TRACKING_INFORMATION_Buffer_1

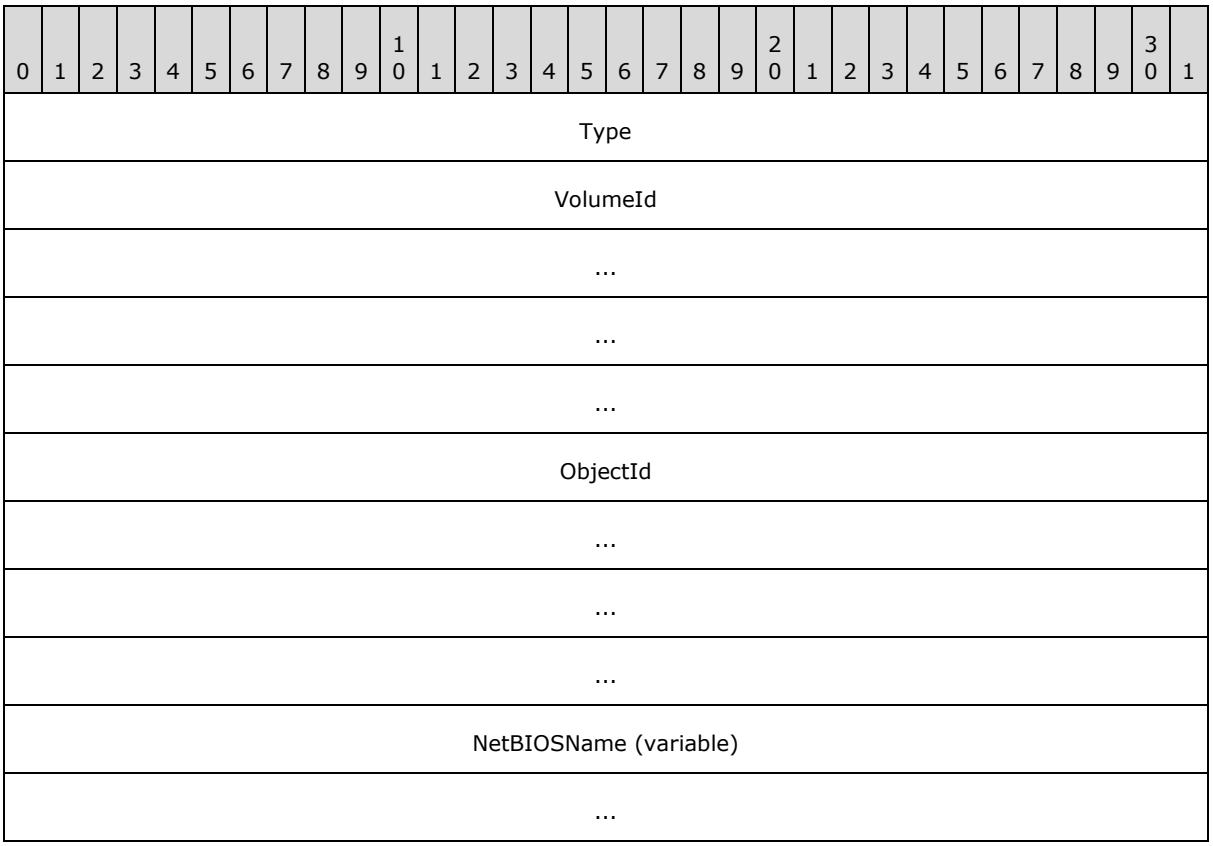
If the **TargetLinkTrackingInformationLength** value is less than 36, the TARGET_LINK_TRACKING_INFORMATION_Buffer data element MUST be as follows.



NetBIOSName (variable): A null-terminated ASCII string containing the NetBIOS name of the destination computer, if known. For more information, see [MS-DLTW] section 3.1.6. If not known, this field is zero length and contains nothing.

2.3.23.3.2 TARGET_LINK_TRACKING_INFORMATION_Buffer_2

If the **TargetLinkTrackingInformationLength** value is greater than or equal to 36, the TARGET_LINK_TRACKING_INFORMATION_Buffer data element MUST be as follows.



Type (4 bytes): An unsigned 32-bit integer that indicates the type of file system on which the file is hosted on the destination computer. MUST be one of the following.

Value	Meaning
0x00000000	The destination file system is NTFS.
0x00000001	The destination file system is DFS. For more information, see [MSDFS].

VolumeId (16 bytes): A 16-byte GUID that uniquely identifies the volume for the object, as obtained from the **ObjectId** field of FileFsObjectIdInformation.

ObjectId (16 bytes): A 16-byte GUID that uniquely identifies the destination file or directory within the volume on which it resides, as indicated by **VolumeId**.

NetBIOSName (variable): A null-terminated ASCII string containing the NetBIOS name of the destination computer, if known. For more information, see [MS-DLTW] section 3.1.6. If not known, this field is zero length and contains nothing.

2.3.24 FSCTL_LMR_SET_LINK_TRACKING_INFORMATION Reply

This message returns the results of the FSCTL_LMR_SET_LINK_TRACKING_INFORMATION request.

The only data item this message returns is a status code, as specified in [MS-ERREF] section 2.3. The status code returned directly by the function that processes this FSCTL MUST be STATUS_SUCCESS or one of the following.

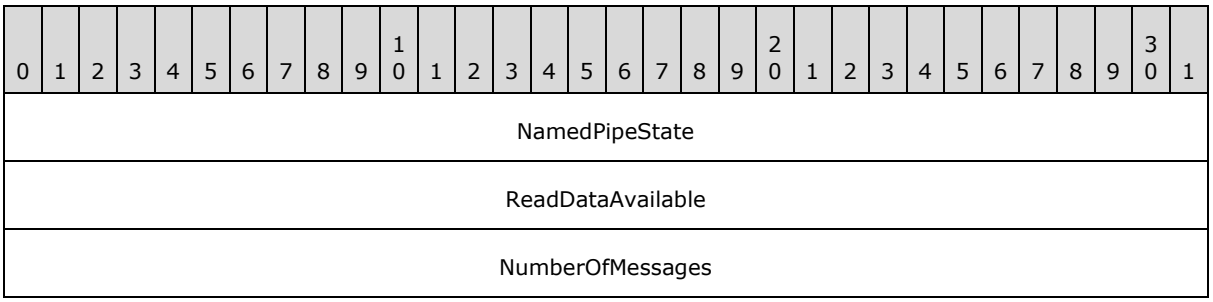
Error code	Meaning
STATUS_INVALID_PARAMETER 0xC000000D	The input buffer length is smaller than the length of the required input data element.

2.3.25 FSCTL_PIPE_PEEK request

The FSCTL_PIPE_PEEK request requests that the server copy a named pipe's data into a buffer for preview without removing it. The FSCTL_PIPE_PEEK request message is issued to invoke a reply, and does not have an associated data structure.

2.3.26 FSCTL_PIPE_PEEK Reply

The **FSCTL_PIPE_PEEK** response returns data from the pipe server's output buffer in the FSCTL output buffer. The structure of that data is as follows.



MessageLength
Data (variable)
...

NamedPipeState (4 bytes): A 32-bit unsigned integer referring to the current state of the pipe. The allowed values are shown in the following table.

Pipe State	Meaning
FILE_PIPE_CONNECTED_STATE 0x00000003	The specified named pipe is in the connected state.
FILE_PIPE_CLOSING_STATE 0x00000004	The server end of the specified named pipe has been closed, but data is still available for the client to read.

ReadDataAvailable (4 bytes): A 32-bit unsigned integer that specifies the size, in bytes, of the data available to read from the pipe.

NumberOfMessages (4 bytes): A 32-bit unsigned integer that specifies the number of messages available in the pipe if the pipe has been created as a message-type pipe. Otherwise, this field is 0.

MessageLength (4 bytes): A 32-bit unsigned integer that specifies the length of the first message available in the pipe if the pipe has been created as a message-type pipe. Otherwise, this field is 0.

Data (variable): A byte buffer of data from the pipe.

This message returns a status code, as specified in [MS-ERREF] section 2.3. The status code returned directly by the function that processes this FSCTL MUST be STATUS_SUCCESS or one of the following.

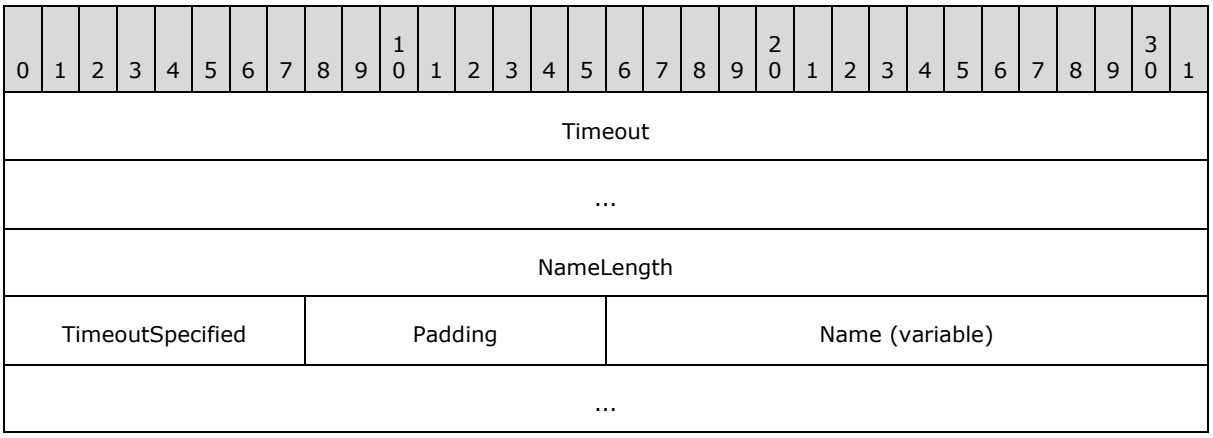
Error code	Meaning
STATUS_PIPE_DISCONNECTED 0xC00000B0	The specified named pipe is in the disconnected state.
STATUS_INVALID_PIPE_STATE 0xC00000AD	The data cannot be read in the current state of the specified pipe.
STATUS_PIPE_BROKEN 0xC000014B	The pipe operation has failed because the other end of the pipe has been closed.
STATUS_INVALID_USER_BUFFER 0xC00000E8	An exception was raised while accessing a user buffer.
STATUS_INSUFFICIENT_RESOURCES 0xC000009A	There were insufficient resources to complete the operation.
STATUS_INVALID_DEVICE_REQUEST 0xC0000010	The type of the handle is not a pipe.

Error code	Meaning
STATUS_BUFFER_OVERFLOW 0x80000005	The data was too large for the specified buffer. This is a warning, not an error. Response contains information including available data length and data that fits into the buffer.

For more information on named pipes, see [PIPE].

2.3.27 FSCTL_PIPE_WAIT Request

The FSCTL_PIPE_WAIT Request requests that the server wait until either a time-out interval elapses or an instance of the specified named pipe is available for connection.



Timeout (8 bytes): A 64-bit signed integer that specifies the maximum amount of time in units of 100 milliseconds that the function can wait for an instance of the named pipe to be available.

NameLength (4 bytes): A 32-bit unsigned integer that specifies the size, in bytes, of the named pipe **Name** field.

TimeoutSpecified (1 byte): A Boolean (section 2.1.8) value that specifies whether or not the **Timeout** parameter will be ignored.

Value	Meaning
FALSE	Indicates that the server MUST wait forever (no timeout) for the named pipe. Any value in **Timeout** MUST be ignored.
TRUE	Indicates that the server MUST use the value in the **Timeout** parameter.

Padding (1 byte): The server MUST ignore this 1-byte padding.

Name (variable): A Unicode string that contains the name of the named pipe. **Name** MUST not include the "\pipe\", so if the operation was on \\server\pipe\pipename, the name would be "pipename".

For more information on named pipes, see [PIPE].

2.3.28 FSCTL_PIPE_WAIT Reply

This message returns the results of the FSCTL_PIPE_WAIT request.

The only data item this message returns is a status code, as specified in [MS-ERREF] section 2.3. The status code returned directly by the function that processes this FSCTL MUST be one of the following.

Error code	Meaning
STATUS_SUCCESS 0x00000000	The specified named pipe is available for connection.
STATUS_OBJECT_NAME_NOT_FOUND 0xC0000034	The specified named pipe does not exist. This error code is also returned when the pipe is closed during wait.
STATUS_IO_TIMEOUT 0xC00000B5	Timeout specified in the FSCTL_PIPE_WAIT request expired.
STATUS_INSUFFICIENT_RESOURCES 0xC000009A	There were insufficient resources to complete the operation.
STATUS_INVALID_DEVICE_REQUEST 0xC0000010	The type of the handle is not a pipe.

2.3.29 FSCTL_PIPE_TRANSCEIVE Request

The FSCTL_PIPE_TRANSCEIVE request is used to send and receive data from an open pipe. Any bytes in the FSCTL input buffer are written as a **binary large object (BLOB)** to the input buffer of the pipe server.

The FSCTL input buffer does not have an associated structure. The buffer is a BLOB of bytes that are written into the associated pipe.

2.3.30 FSCTL_PIPE_TRANSCEIVE Reply

The FSCTL_PIPE_TRANSCEIVE response returns data from the pipe server's output buffer in the FSCTL output buffer.

This message returns a status code, as specified in [MS-ERREF] section 2.3. The status code returned directly by the function that processes this FSCTL MUST be STATUS_SUCCESS or one of the following.

Error code	Meaning
STATUS_PIPE_DISCONNECTED 0xC00000B0	The specified named pipe is in the disconnected state.
STATUS_INVALID_PIPE_STATE 0xC00000AD	The named pipe is not in the connected state or not in the full-duplex message mode.
STATUS_PIPE_BUSY 0xC00000AE	The named pipe contains unread data.

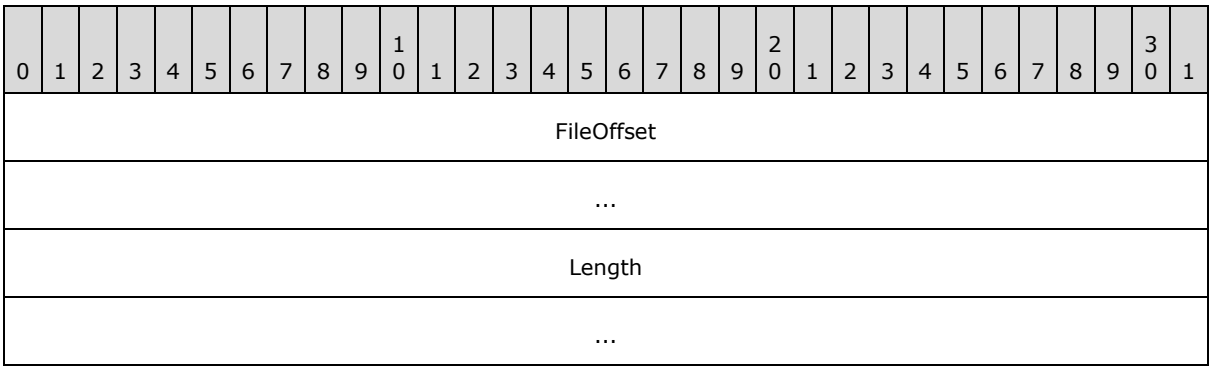
Error code	Meaning
STATUS_INVALID_USER_BUFFER 0xC00000E8	An exception was raised while accessing a user buffer.
STATUS_INSUFFICIENT_RESOURCES 0xC000009A	There were insufficient resources to complete the operation.
STATUS_INVALID_DEVICE_REQUEST 0xC0000010	The type of the handle is not a pipe.
STATUS_BUFFER_OVERFLOW 0x80000005	The data was too large to fit into the specified buffer.

For more information on named pipes, see [PIPE].

2.3.31 FSCTL_QUERY_ALLOCATED_RANGES Request

The FSCTL_QUERY_ALLOCATED_RANGES request message requests that the server scan a file or alternate stream looking for byte ranges that may contain nonzero data, and then return information on those ranges. Only sparse files can have zeroed ranges known to the operating system. For other files, the server will return only a single range that contains the starting point and the length requested. The request message contains a FILE_ALLOCATED_RANGE_BUFFER data element.

The FILE_ALLOCATED_RANGE_BUFFER data element is as follows.



FileOffset (8 bytes): A 64-bit signed integer that contains the file offset, in bytes, of the start of a range of bytes in a file. The value of this field MUST be greater than or equal to 0.

Length (8 bytes): A 64-bit signed integer that contains the size, in bytes, of the range. In a request message, the value of this field MUST be greater than or equal to 0. In a reply message, it MUST be greater than 0.

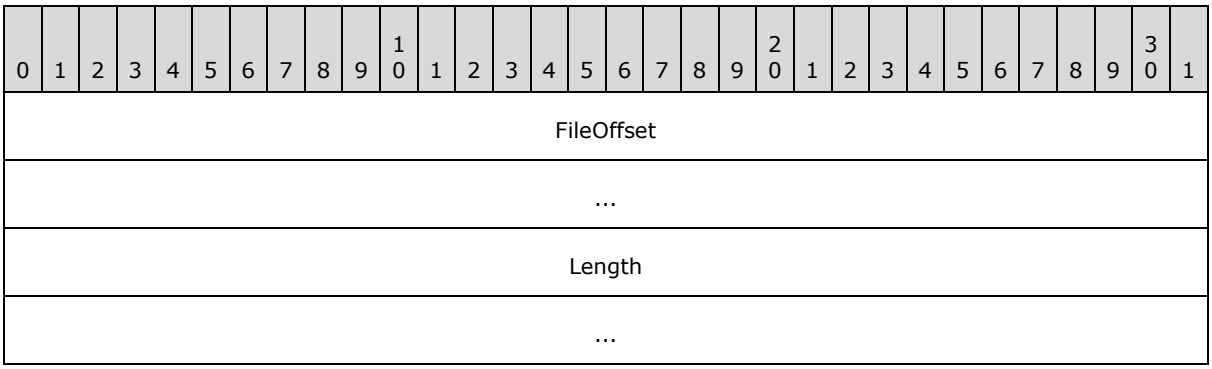
2.3.32 FSCTL_QUERY_ALLOCATED_RANGES Reply

The FSCTL_QUERY_ALLOCATED_RANGES Reply message returns the results of the FSCTL_QUERY_ALLOCATED_RANGES Request (section 2.3.31).

This message MUST return an array of zero or more FILE_ALLOCATED_RANGE_BUFFER data elements. The number of FILE_ALLOCATED_RANGE_BUFFER elements returned is computed by dividing the size of the returned output buffer (from either SMB or SMB2, the lower-layer protocol

that carries the FSCTL) by the size of the FILE_ALLOCATED_RANGE_BUFFER element. Ranges returned MUST intersect the range specified in the FSCTL_QUERY_ALLOCATED_RANGES Request. Zero FILE_ALLOCATED_RANGE_BUFFER data elements MUST be returned when the file has no allocated ranges.<22>

The FILE_ALLOCATED_RANGE_BUFFER data element is as follows.



FileOffset (8 bytes): A 64-bit signed integer that contains the file offset in bytes from the start of the file; the start of a range of bytes to which storage is allocated. If the file is a sparse file, it can contain ranges of bytes for which storage is not allocated; these ranges will be excluded from the list of allocated ranges returned by this FSCTL.<23> Because an application using a sparse file can choose whether or not to allocate disk space for each sequence of 0x00-valued bytes, the allocated ranges can contain 0x00-valued bytes. This value MUST be greater than or equal to 0.<24>

Length (8 bytes): A 64-bit signed integer that contains the size, in bytes, of the range. In a request message, the value of this field MUST be greater than or equal to 0. In a reply message, it MUST be greater than 0.

This message returns a status code, as specified in [MS-ERREF] section 2.3. The status code returned directly by the function that processes this FSCTL MUST be STATUS_SUCCESS or one of the following.

Error code	Meaning
STATUS_INVALID_PARAMETER 0xC000000D	The handle is not to a file, or the size of the input buffer is less than the size of a FILE_ALLOCATED_RANGE_BUFFER structure, or the given **FileOffset** field value is less than zero, or the given **Length** field value is less than zero, or the given **FileOffset** field value plus the given **Length** field value is larger than 0x7FFFFFFFFFFFFFFF.
STATUS_INVALID_USER_BUFFER 0xC00000E8	The input buffer or output buffer is not aligned to a 4-byte boundary.
STATUS_BUFFER_TOO_SMALL 0xC0000023	The output buffer is too small to contain a FILE_ALLOCATED_RANGE_BUFFER structure.
STATUS_BUFFER_OVERFLOW 0x80000005	The output buffer is too small to contain the required number of FILE_ALLOCATED_RANGE_BUFFER structures.

2.3.33 FSCTL_QUERY_FAT_BPB Request

This message requests that the server return the first 0x24 bytes of sector 0 for the volume that contains the file or directory associated with the handle on which this FSCTL was invoked. The first 0x24 bytes of sector 0 are known as the FAT BIOS Parameter Block (BPB), which contains hardware-specific bootstrap information.

This message does not contain any additional data elements.

This FSCTL is valid only for a FAT file system. All other file systems treat this as an invalid FSCTL.

2.3.34 FSCTL_QUERY_FAT_BPB Reply

The reply buffer contains the first 0x24 bytes of sector 0 for the volume associated with the handle on which this FSCTL was invoked.

This message also returns a status code as specified in [MS-ERREF] section 2.3. The status code returned directly by the function that processes this FSCTL MUST be STATUS_SUCCESS or one of the following.

Error Code	Meaning
STATUS_INVALID_DEVICE_REQUEST 0xC0000010	The specified request is not a valid operation for the target device.
STATUS_BUFFER_TOO_SMALL 0xC0000023	The buffer is too small to contain the entry. No information has been written to the buffer.

2.3.35 FSCTL_QUERY_ON_DISK_VOLUME_INFO Request

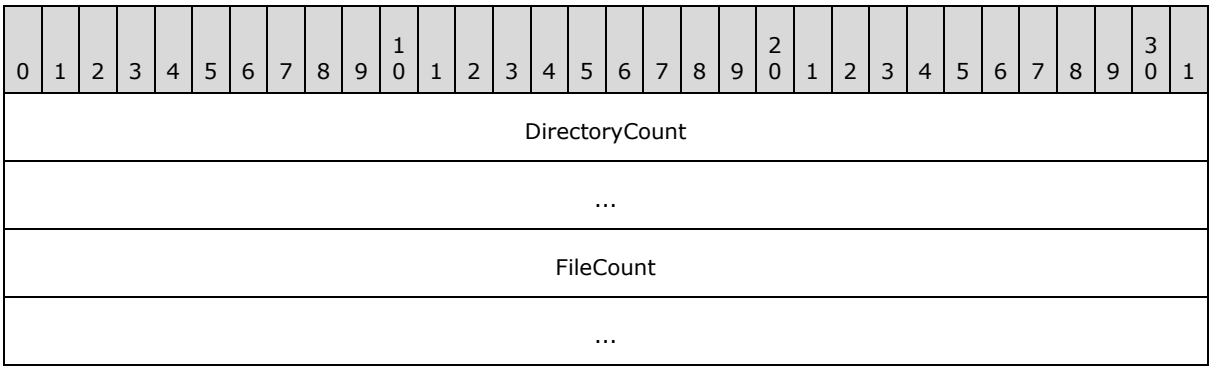
This message requests UDF-specific volume information for the volume that contains the file or directory associated with the handle on which this FSCTL was invoked.

This message does not contain any additional data elements.

This FSCTL is only valid on UDF file systems. All other File Systems will treat this as an invalid FSCTL. For information regarding UDF, see [UDF].

2.3.36 FSCTL_QUERY_ON_DISK_VOLUME_INFO Reply

This message returns the results of the FSCTL_QUERY_ON_DISK_VOLUME_INFO request (section 2.3.35).



FsFormatMajVersion	FsFormatMinVersion
FsFormatName	
...	
...	
...	
...	
...	
FormatTime	
...	
LastUpdateTime	
...	
CopyrightInfo	
...	
...	
...	
...	
...	
...	
...	
(CopyrightInfo cont'd for 9 rows)	
AbstractInfo	
...	
...	

...
...
...
...
...
(AbstractInfo cont'd for 9 rows)
FormattingImplementationInfo
...
...
...
...
...
...
...
(FormattingImplementationInfo cont'd for 9 rows)
LastModifyingImplementationInfo
...
...
...
...
...
...
...

(LastModifyingImplementationInfo cont'd for 9 rows)

DirectoryCount (8 bytes): A 64-bit signed integer. The number of directories on the specified volume. This member is -1 if the number is unknown.

For UDF file systems with a virtual allocation table, this information is available only if the UDF revision of the volume is greater than 1.50.<25>

FileCount (8 bytes): A 64-bit signed integer. The number of files on the specified volume. Returns -1 if the number is unknown.

For UDF file systems with a virtual allocation table, this information is available only if the UDF revision of the volume is greater than 1.50.

FsFormatMajVersion (2 bytes): A 16-bit signed integer. The major version number of the file system. Returns -1 if the number is unknown or not applicable. For example on UDF 1.02 file systems, 1 is returned.

FsFormatMinVersion (2 bytes): A 16-bit signed integer. The minor version number of the file system. Returns -1 if the number is unknown or not applicable. For example: on UDF 1.02 file systems, 2 is returned.

FsFormatName (24 bytes): Always returns "UDF" in Unicode characters followed by nine Unicode NULL characters.

FormatTime (8 bytes): The time the volume was formatted; see section 2.1.1.

LastUpdateTime (8 bytes): The time the volume was last updated; see section 2.1.1.

CopyrightInfo (68 bytes): A Unicode string containing any copyright notifications associated with the volume. This information is implementation-specific and will be padded with NULLs.<26>

AbstractInfo (68 bytes): A Unicode string containing any abstract information written on the volume. This information is implementation-specific and will be padded with NULLs.<27>

FormattingImplementationInfo (68 bytes): A Unicode string containing the operating system version that the volume was formatted by. This information is implementation-specific and will be padded with NULLs.<28>

LastModifyingImplementationInfo (68 bytes): A Unicode string containing the operating system version that the volume was last modified by. This information is implementation-specific and will be padded with NULLs.<29>

This message returns a status code, as specified in [MS-ERREF] section 2.3. The status code returned directly by the function that processes this FSCTL MUST be STATUS_SUCCESS or one of the following.

Error Code	Meaning
STATUS_INVALID_USER_BUFFER 0xC00000E8	An access to a user buffer failed.
STATUS_BUFFER_TOO_SMALL 0xC0000023	The buffer is too small to contain the entry. No information has been written to the buffer.

Error Code	Meaning
STATUS_INVALID_PARAMETER 0xC000000D	An invalid parameter was passed to a service or function.

2.3.37 FSCTL_QUERY_SPARING_INFO Request

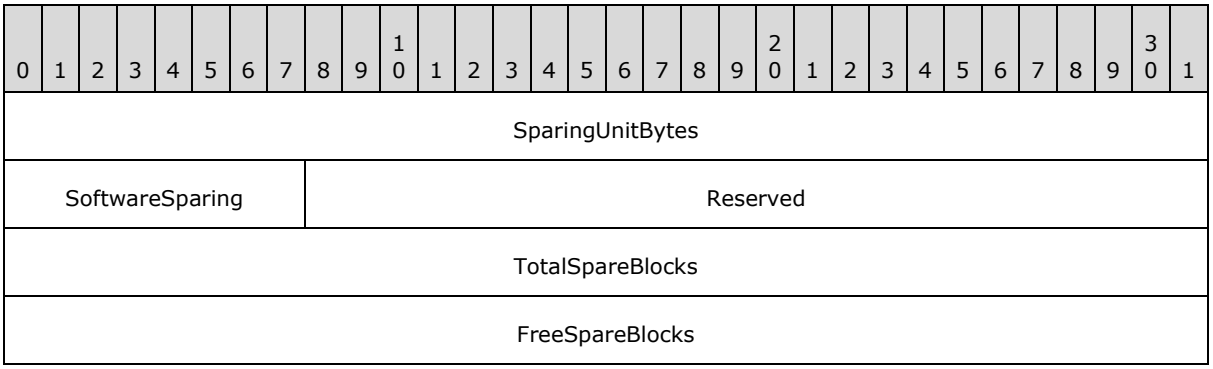
Retrieves the defect management properties of the volume that contains the file or directory associated with the handle on which this FSCTL was invoked.

This message does not contain any additional data elements.

This FSCTL is only valid on UDF file systems. All other file systems will treat this as an invalid FSCTL. For information regarding UDF, see [UDF].

2.3.38 FSCTL_QUERY_SPARING_INFO Reply

This message returns the results of the FSCTL_QUERY_SPARING_INFO request (section 2.3.37).



SparingUnitBytes (4 bytes): The size of a sparing packet and the underlying error check and correction (ECC) block size of the volume.

SoftwareSparing (1 byte): A Boolean (section 2.1.8) value. If TRUE, indicates that sparing behavior is software-based; if FALSE, it is hardware-based.

Reserved (3 bytes): A 24-bit reserved value. This field SHOULD be set to zero, and MUST be ignored.

TotalSpareBlocks (4 bytes): The total number of blocks allocated for sparing.

FreeSpareBlocks (4 bytes): The number of blocks available for sparing.

This message returns a status code, as specified in [MS-ERREF] section 2.3. The status code returned directly by the function that processes this FSCTL MUST be STATUS_SUCCESS or the following.

Error code	Meaning
STATUS_INVALID_PARAMETER 0xC000000D	An invalid parameter was passed to a service or function, or the buffer is too small to contain the entry.

2.3.39 FSCTL_READ_FILE_USN_DATA Request

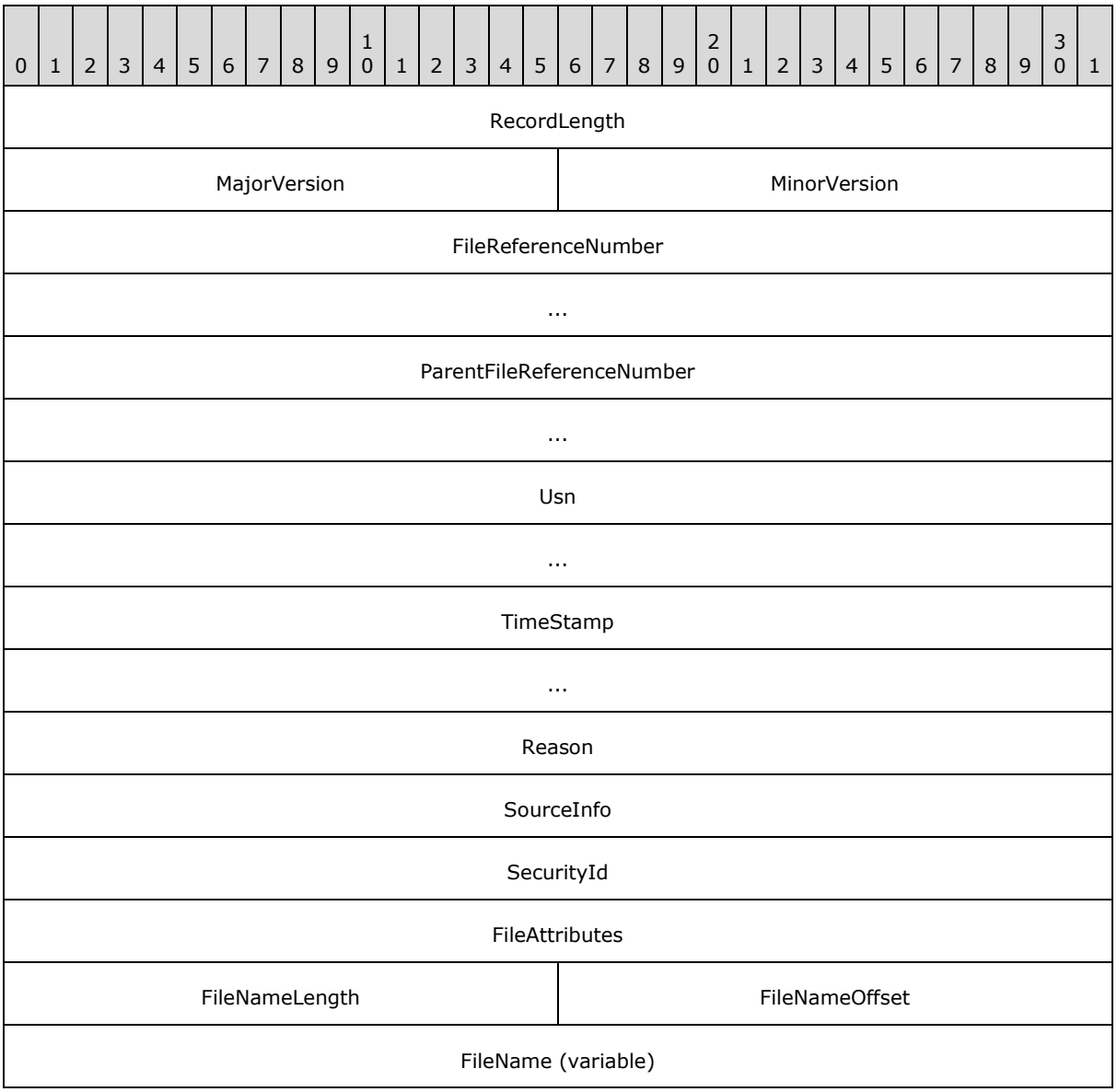
This message requests that the server return the most recent change journal **USN** for the file or directory associated with the handle on which this FSCTL was invoked.

This message does not contain any additional data elements.

2.3.40 FSCTL_READ_FILE_USN_DATA Reply

The FSCTL_READ_FILE_USN_DATA reply message returns the results of the FSCTL_READ_FILE_USN_DATA request as a USN_RECORD.

The USN_RECORD element is as follows.



...

RecordLength (4 bytes): A 32-bit unsigned integer that contains the total length of the update sequence number (USN) record, in bytes.

MajorVersion (2 bytes): A 16-bit unsigned integer that contains the major version of the change journal software for this record. For example, if the change journal software is version 2.0, the major version number is 2.<30>

MinorVersion (2 bytes): A 16-bit unsigned integer that contains the minor version of the change journal software for this record. For example, if the change journal software is version 2.0, the minor version number is 0 (zero).<31>

FileReferenceNumber (8 bytes): A 64-bit unsigned integer, opaque to the client, containing the number (assigned by the file system when the file is created) of the file or directory for which this record notes changes. The FileReferenceNumber is an arbitrarily assigned value (unique within the volume on which the file is stored) that associates a journal record with a file. This value SHOULD always be unique within the volume on which the file is stored over the life of the volume.<32>

ParentFileReferenceNumber (8 bytes): A 64-bit unsigned integer, opaque to the client, containing the ordinal number of the directory on which the file or directory that is associated with this record is located. This is an arbitrarily assigned value (unique within the volume on which the file is stored) that associates a journal record with a parent directory.

Usn (8 bytes): A 64-bit signed integer, opaque to the client, containing the USN of the record. This value is unique within the volume on which the file is stored. This value MUST be greater than or equal to 0. This value MUST be 0 if no USN change journal records have been logged for the file or directory associated with this record. For more information, see [MSDN-CJ].

TimeStamp (8 bytes): The absolute system time that this change journal event was logged; see section 2.1.1.

Reason (4 bytes): A 32-bit unsigned integer that contains flags that indicate reasons for changes that have accumulated in this file or directory journal record since the file or directory was opened. When a file or directory is closed, a final USN record is generated with the USN_REASON_CLOSE flag set in this field. The next change, occurring after the next open operation or deletion, starts a new record with a new set of reason flags. A rename or move operation generates two USN records: one that records the old parent directory for the item and one that records the new parent in the **ParentFileReferenceNumber** member. Possible values for the reason code are as follows (all unused bits are reserved for future use and MUST NOT be used).

Value	Meaning
USN_REASON_BASIC_INFO_CHANGE 0x00008000	A user has either changed one or more files or directory attributes (such as read-only, hidden, archive, or sparse) or one or more time stamps.
USN_REASON_CLOSE 0x80000000	The file or directory is closed.
USN_REASON_COMPRESSION_CHANGE 0x00020000	The compression state of the file or directory is changed from (or to) compressed.

Value	Meaning
USN_REASON_DATA_EXTEND 0x00000002	The file or directory is extended (added to).
USN_REASON_DATA_OVERWRITE 0x00000001	The data in the file or directory is overwritten.
USN_REASON_DATA_TRUNCATION 0x00000004	The file or directory is truncated.
USN_REASON_EA_CHANGE 0x00000400	The user made a change to the extended attributes of a file or directory. These NTFS file system attributes are not accessible to nonnative applications. This USN reason does not appear under normal system usage, but can appear if an application or utility bypasses the Win32 API and uses the native API to create or modify extended attributes of a file or directory.
USN_REASON_ENCRYPTION_CHANGE 0x00040000	The file or directory is encrypted or decrypted.
USN_REASON_FILE_CREATE 0x00000100	The file or directory is created for the first time.
USN_REASON_FILE_DELETE 0x00000200	The file or directory is deleted.
USN_REASON_HARD_LINK_CHANGE 0x00010000	A hard link is added to (or removed from) the file or directory.
USN_REASON_INDEXABLE_CHANGE 0x00004000	A user changes the FILE_ATTRIBUTE_NOT_CONTEXT_INDEXED attribute. That is, the user changes the file or directory from one in which content can be indexed to one in which content cannot be indexed, or vice versa.
USN_REASON_NAMED_DATA_EXTEND 0x00000020	The one (or more) named data stream for a file is extended (added to).
USN_REASON_NAMED_DATA_OVERWRITE 0x00000010	The data in one (or more) named data stream for a file is overwritten.
USN_REASON_NAMED_DATA_TRUNCATION 0x00000040	One (or more) named data stream for a file is truncated.
USN_REASON_OBJECT_ID_CHANGE 0x00080000	The object identifier of a file or directory is changed.
USN_REASON_RENAME_NEW_NAME 0x00002000	A file or directory is renamed, and the file name in the USN_RECORD structure is the new name.
USN_REASON_RENAME_OLD_NAME 0x00001000	The file or directory is renamed, and the file name in the USN_RECORD structure is the previous name.
USN_REASON_REPARSE_POINT_CHANGE 0x00100000	The reparse point that is contained in a file or directory is changed, or a reparse point is added to

Value	Meaning
	(or deleted from) a file or directory.
USN_REASON_SECURITY_CHANGE 0x00000800	A change is made in the access rights to a file or directory.
USN_REASON_STREAM_CHANGE 0x00200000	A named stream is added to (or removed from) a file, or a named stream is renamed.

SourceInfo (4 bytes): A 32-bit unsigned integer that provides additional information about the source of the change. When a thread writes a new USN record, the source information flags in the prior record continue to be present only if the thread also sets those flags. Therefore, the source information structure allows applications to filter out USN records that are set only by a known source, for example, an antivirus filter. This flag MUST contain one of the following values.

Value	Meaning
USN_SOURCE_DATA_MANAGEMENT 0x00000001	The operation provides information about a change to the file or directory that was made by the operating system. For example, a change journal record with this SourceInfo value is generated when the Remote Storage system moves data from external to local storage. This SourceInfo value indicates that the modifications did not change the application data in the file.
USN_SOURCE_AUXILIARY_DATA 0x00000002	The operation adds a private data stream to a file or directory. For example, a virus detector might add checksum information. As the virus detector modifies the item, the system generates USN records. This SourceInfo value indicates that the modifications did not change the application data in the file.
USN_SOURCE_REPLICATION_MANAGEMENT 0x00000004	The operation modified the file to match the content of the same file that exists in another member of the replica set for the File Replication Service (FRS).

SecurityId (4 bytes): A 32-bit unsigned integer that contains an index of a unique security identifier assigned to the file or directory associated with this record. This index is internal to the underlying object store and MUST be ignored.

FileAttributes (4 bytes): A 32-bit unsigned integer that contains attributes for the file or directory associated with this record. Attributes of streams associated with the file or directory are excluded. Valid file attributes are specified in section 2.6.

FileNameLength (2 bytes): A 16-bit unsigned integer that contains the length of the file or directory name associated with this record, in bytes. The **FileName** member contains this name. Use this member to determine file name length rather than depending on a trailing null to delimit the file name in **FileName**.

FileNameOffset (2 bytes): A 16-bit unsigned integer that contains the offset, in bytes, of the **FileName** member from the beginning of the structure.

FileName (variable): A variable-length field of Unicode characters containing the name of the file or directory associated with this record in Unicode format. When working with this field, do not assume that the file name will contain a trailing Unicode null character.

The fields **Reason**, **TimeStamp**, **SourceInfo**, and **SecurityId** for a USN RECORD element returned by this FSCTL MUST all be set 0.<33>

This message returns a status code, as specified in [MS-ERREF] section 2.3. The status code returned directly by the function that processes this FSCTL MUST be STATUS_SUCCESS or one of the following.

Error code	Meaning
STATUS_INVALID_PARAMETER 0xC000000D	The handle is not to a file or directory.
STATUS_INVALID_USER_BUFFER 0xC00000E8	The output buffer is not aligned to a 4-byte boundary.
STATUS_BUFFER_TOO_SMALL 0xC0000023	The output buffer is too small to contain a USN_RECORD structure.
STATUS_INVALID_DEVICE_REQUEST 0xC0000010	The file system does not support the use of a USN change journal.

2.3.41 FSCTL_RECALL_FILE Request

This message requests that the server recall the file (associated with the handle on which this FSCTL was invoked) from storage media that Remote Storage manages. This FSCTL is not valid for directories.

Typically, files stored on media that is managed by Remote Storage are recalled when an application attempts to make the first access to data. An application that opens a file without immediately accessing the data can speed up the first access by using FSCTL_RECALL_FILE immediately after opening the file. For performance reasons, an application should not recall a file unnecessarily.

This message does not contain any additional data elements.

2.3.42 FSCTL_RECALL_FILE Reply

This message returns the results of the FSCTL_RECALL_FILE request.

The only data item this message returns is a status code, as specified in [MS-ERREF] section 2.3. The status code returned directly by the function that processes this FSCTL MUST be STATUS_SUCCESS or one of the following.

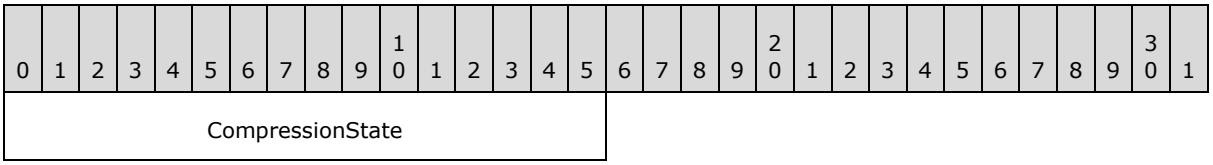
Error code	Meaning
STATUS_ACCESS_DENIED 0xC0000022	The file is set to not allow recall.
ERROR_INVALID_FUNCTION 0x00000001	The Remote Storage option is not installed.
STATUS_NOT_SUPPORTED	The request is not supported.

Error code	Meaning
0xC00000BB	
STATUS_INVALID_DEVICE_REQUEST 0xC0000010	The supplied handle is not that of a file.

2.3.43 FSCTL_SET_COMPRESSION Request

The FSCTL_SET_COMPRESSION request message requests that the server set the compression state of the file or directory associated with the handle on which this FSCTL was invoked. The message contains a 16-bit unsigned integer.

The CompressionState element is as follows.



CompressionState (2 bytes): MUST be one of the following standard values.

Value	Meaning
COMPRESSION_FORMAT_NONE 0x0000	The file or directory is not compressed.
COMPRESSION_FORMAT_DEFAULT 0x0001	The file or directory is compressed by using the default compression algorithm.<34>
COMPRESSION_FORMAT_LZNT1 0x0002	The file or directory is compressed by using the LZNT1 compression algorithm. For more information, see [UASDC].
All other values	Reserved for future use and MUST NOT be used.

The actual file or directory compression performed when a server receives a request for COMPRESSION_FORMAT_DEFAULT and COMPRESSION_FORMAT_LZNT1 is implementation-dependent.<35>

If the file system of the volume containing the specified file or directory does not support per-file or per-directory compression, the request MUST NOT succeed. The error code returned in this situation is specified in section 2.2.

2.3.44 FSCTL_SET_COMPRESSION Reply

This message returns the results of the FSCTL_SET_COMPRESSION request.

The only data item this message returns is a status code, as specified in [MS-ERREF] section 2.3. The status code returned directly by the function that processes this FSCTL MUST be STATUS_SUCCESS or one of the following.

Error code	Meaning
STATUS_INVALID_PARAMETER	The input buffer length is less than 2, or the handle is not to a file

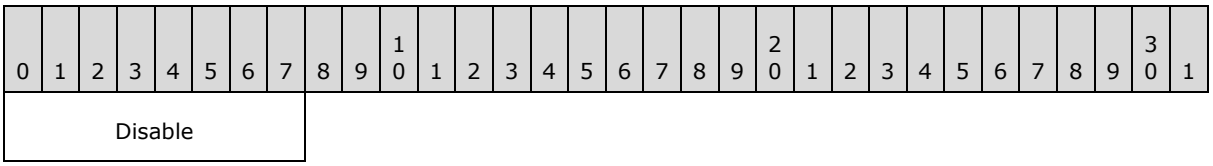
Error code	Meaning
0xC000000D	or directory, or the requested CompressionState is not one of the values listed in the table for **CompressionState** in FSCTL_SET_COMPRESSION Request (section 2.3.43).
STATUS_INVALID_DEVICE_REQUEST 0xC0000010	The volume does not allow compression.
STATUS_DISK_FULL 0xC000007	The disk is full.

2.3.45 FSCTL_SET_DEFECT_MANAGEMENT Request

Sets the software defect management state for the specified file associated with the handle on which this FSCTL was invoked. Used for UDF file systems.

This message contains a **FILE_SET_DEFECT_MGMT_BUFFER** structure.

FILE_SET_DEFECT_MGMT_BUFFER is defined as follows.



Disable (1 byte): A Boolean (section 2.1.8) value. If TRUE, indicates that defect management will be disabled. If FALSE, indicates that defect management will be enabled.

This FSCTL is valid only on UDF file systems. All other file systems will treat this as an invalid FSCTL. For information regarding UDF, see [UDF].

2.3.46 FSCTL_SET_DEFECT_MANAGEMENT Reply

The only data item this message returns is a status code, as specified in [MS-ERREF] section 2.3. The status code returned directly by the function that processes this FSCTL MUST be STATUS_SUCCESS or one of the following.

Error code	Meaning
STATUS_INVALID_PARAMETER 0xC000000D	An invalid parameter was passed to a service or function or the handle on which this FSCTL was invoked is that of a directory.
STATUS_INVALID_DEVICE_REQUEST 0xC0000010	The specified request is not a valid operation for the target device.
STATUS_SHARING_VIOLATION 0xC0000043	A file cannot be opened because the share access flags are incompatible.
STATUS_VOLUME_DISMOUNTED 0xC000026e	An operation was attempted to a volume after it was dismounted.
STATUS_FILE_INVALID 0xC0000098	The volume for a file has been externally altered such that the opened file is no longer valid.

Error code	Meaning
STATUS_WRONG_VOLUME 0xC0000012	The wrong volume is in the drive.
STATUS_VERIFY_REQUIRED 0x80000016	The media has changed and a verify operation is in progress so no reads or writes may be performed to the device, except those used in the verify operation.

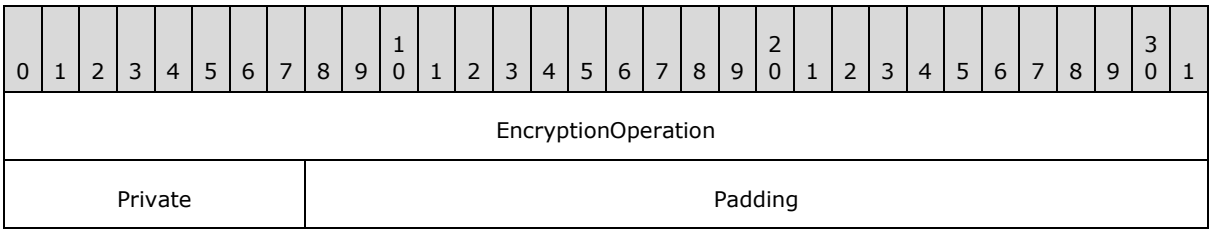
There are no additional data elements in this reply.

2.3.47 FSCTL_SET_ENCRYPTION Request

The FSCTL_SET_ENCRYPTION request sets the encryption for the file or directory associated with the given handle.<36><37>

The message contains an ENCRYPTION_BUFFER structure that indicates whether to encrypt/decrypt a file or an individual stream.

ENCRYPTION_BUFFER is defined as follows.



EncryptionOperation (4 bytes): A 32-bit unsigned integer value that indicates the operation to be performed. The valid values are as follows.

Value	Meaning
FILE_SET_ENCRYPTION 0x00000001	This operation requests encryption of the specified file or directory.<38>
FILE_CLEAR_ENCRYPTION 0x00000002	This operation requests removal of encryption from the specified file or directory. It MUST fail if any streams for the file are marked encrypted.<39>
STREAM_SET_ENCRYPTION 0x00000003	This operation requests encryption of the specified stream.<40>
STREAM_CLEAR_ENCRYPTION 0x00000004	This operation requests the removal of encryption from the specified stream.<41>

Private (1 byte): An 8-bit unsigned char value.<42>

Padding (3 bytes): These bytes MUST be ignored.

2.3.48 FSCTL_SET_ENCRYPTION Reply

This message returns the results of the FSCTL_SET_ENCRYPTION request. If the file system of the volume containing the specified file or directory does not support encryption, the request MUST NOT succeed. The error code returned in this situation varies, depending on the file system.

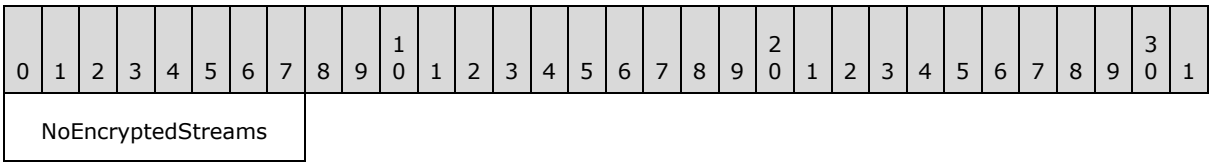
This message returns a status code, as specified in [MS-ERREF] section 2.3, as well as a DECRYPTION_STATUS_BUFFER (section 2.3.48.1) if an output buffer is passed in.

The status code returned directly by the function that processes this FSCTL MUST be STATUS_SUCCESS<43> or one of the following.

Error code	Meaning
STATUS_MEDIA_WRITE_PROTECTED 0xC00000A2	The disk cannot be written to because it is write-protected.
STATUS_INVALID_PARAMETER 0xC000000D	The **EncryptionOperation** field value is invalid, the open request is not for a file or directory or stream encryption has been requested on a stream that is compressed.
STATUS_BUFFER_TOO_SMALL 0xC0000023	The size of the input buffer is less than the size of the encryption buffer structure defined in section 2.3.47, or an output buffer is present and is smaller than a DECRYPTION_STATUS_BUFFER structure.
STATUS_VOLUME_NOT_UPGRADED 0xC000029C	The version of the file system on the volume does not support encryption.<44>
STATUS_INVALID_DEVICE_REQUEST 0xC0000010	The request was invalid for a system-specific reason. <45>
STATUS_FILE_CORRUPT_ERROR 0xC0000102	A required attribute is missing from a directory for which encryption was requested. <46>
STATUS_VOLUME_DISMOUNTED 0xC000026E	The volume is not mounted.
STATUS_INVALID_USER_BUFFER 0xC00000E8	An exception was raised while accessing a user buffer.

2.3.48.1 DECRYPTION_STATUS_BUFFER

The DECRYPTION_STATUS_BUFFER is defined as follows.



NoEncryptedStreams (1 byte): A Boolean (section 2.1.8) value. A TRUE value means that the last encrypted stream of the specified file was just decrypted by an FSCTL_SET_ENCRYPTION operation; otherwise, a FALSE value is returned.

2.3.49 FSCTL_SET_OBJECT_ID Request

This message sets the object identifier for the file or directory associated with the handle on which this FSCTL was invoked. The message contains a FILE_OBJECTID_BUFFER (section 2.1.3) data element. Either a Type 1 or a Type 2 buffer is valid.<47><48>

2.3.50 FSCTL_SET_OBJECT_ID Reply

This message returns the results of the FSCTL_SET_OBJECT_ID request.

If the file system of the volume containing the specified file or directory does not support the use of object IDs, the request will not succeed. The error code returned in this situation varies, depending on the file system.

The only data item this message returns is a status code, as specified in [MS-ERREF] section 2.3. The status code returned directly by the function that processes this FSCTL MUST be STATUS_SUCCESS or one of the following.

Error code	Meaning
STATUS_INVALID_PARAMETER 0xC000000D	The handle is not to a file or directory, or the input buffer's length is not equal to the size of a FILE_OBJECTID_BUFFER structure.
STATUS_ACCESS_DENIED 0xC0000022	The handle was not opened with write data or write attribute access as well as restore access.
STATUS_OBJECT_NAME_COLLISION 0xC0000035	The file or directory already has an object ID.
STATUS_INVALID_DEVICE_REQUEST 0xC0000010	The file system does not support the use of object IDs.
STATUS_MEDIA_WRITE_PROTECTED 0xC00000A2	The volume is write-protected and changes to it cannot be made.

2.3.51 FSCTL_SET_OBJECT_ID_EXTENDED Request

The FSCTL_SET_OBJECT_ID_EXTENDED request message requests that the server set the extended information for the file or directory associated with the handle on which this FSCTL was invoked. The message contains an EXTENDED_INFO data element.

The EXTENDED_INFO data element is defined as follows.

0	1	2	3	4	5	6	7	8	9	10	1	2	3	4	5	6	7	8	9	20	1	2	3	4	5	6	7	8	9	30	1
ExtendedInfo																															
...																															
...																															
...																															

...
...
...
...
(ExtendedInfo cont'd for 4 rows)

ExtendedInfo (48 bytes): A 48-byte binary large object(BLOB) containing user-defined extended data that was passed to this FSCTL by an application. In this situation, the user refers to the implementer who is calling this FSCTL, meaning the extended info is opaque to NTFS; there are no rules enforced by NTFS as to what these last 48 bytes contain. Contrast this with the first 16 bytes of an object ID, which can be used to open the file, so NTFS requires that they be unique within a volume.<49>

2.3.52 FSCTL_SET_OBJECT_ID_EXTENDED Reply

This message returns the results of the FSCTL_SET_OBJECT_ID_EXTENDED request.

If the file system of the volume containing the specified file or directory does not support the use of ObjectIds, the request will not succeed. The error code returned in this situation varies, depending on the file system.

The only data item this message returns is a status code, as specified in [MS-ERREF] section 2.3. The status code returned directly by the function that processes this FSCTL MUST be STATUS_SUCCESS or one of the following.

Error code	Meaning
STATUS_INVALID_PARAMETER 0xC000000D	The handle is not to a file or directory, or the input buffer's length is not equal to the size of an EXTENDED_INFO structure.
STATUS_ACCESS_DENIED 0xC0000022	The handle was not opened with write data or write attribute access.
STATUS_OBJECT_NAME_NOT_FOUND 0xC0000034	The file or directory has no object ID.
STATUS_INVALID_DEVICE_REQUEST 0xC0000010	The file system does not support the use of object IDs.

2.3.53 FSCTL_SET_REPARSE_POINT Request

This message requests that the server set a reparse point on the file or directory associated with the handle on which this FSCTL was invoked.

The message contains a REPARSE_GUID_DATA_BUFFER or a REPARSE_DATA_BUFFER (including subtypes) data element. Both the REPARSE_GUID_DATA_BUFFER and REPARSE_DATA_BUFFER structures begin with a **ReparseTag** field. The ReparseTag value uniquely identifies the filter driver that creates/uses the reparse point, and the filter driver processes the reparse point data as either a

REPARSE_GUID_DATA_BUFFER or a REPARSE_DATA_BUFFER, depending on the structure implemented by the filter driver for that type of reparse point.

This message is applicable only to a file or directory handle, not to a volume handle.

2.3.54 FSCTL_SET_REPARSE_POINT Reply

This message returns the results of the FSCTL_SET_REPARSE_POINT request.

If the file system of the volume containing the specified file or directory does not support reparse points, the request will not succeed. The error code returned in this situation varies, depending on the file system.

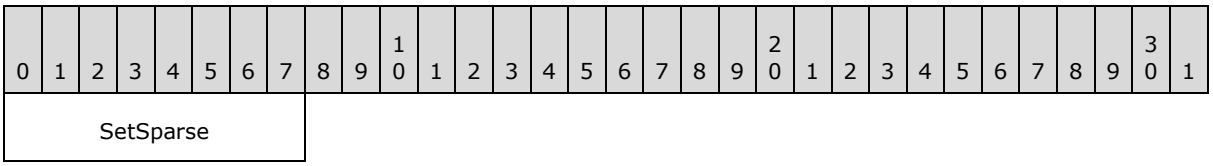
The only data item this message returns is a status code, as specified in [MS-ERREF] section 2.3. The status code returned directly by the function that processes this FSCTL MUST be STATUS_SUCCESS or one of the following.

Error code	Meaning
STATUS_INVALID_PARAMETER 0xC000000D	The handle is not to a file or directory, or the output buffer's length is greater than 0.
STATUS_IO_REPARSE_DATA_INVALID 0xC0000278	The input buffer length is less than the size of a REPARSE_DATA_BUFFER structure, or the input buffer length is greater than 16,384, or a REPARSE_DATA_BUFFER structure has been specified for a third party reparse tag, or the GUID specified for a third party reparse tag does not match the GUID known by the operating system for this reparse point, or the reparse tag is 0 or 1.
STATUS_INVALID_DEVICE_REQUEST 0xC0000010	The file system does not support reparse points.

2.3.55 FSCTL_SET_SPARSE Request

This message requests that the server mark the file that is associated with the handle on which this FSCTL was invoked as sparse. In a sparse file, large ranges of zeros (0) may not require disk allocation. Space for nonzero data is allocated as the file is written. The message either has no data elements at all or it contains a FILE_SET_SPARSE_BUFFER element. If there is no data element, the sparse flag for the file is set, exactly as if the FILE_SET_SPARSE_BUFFER element was supplied and had a **SetSparse** value of TRUE.

The FILE_SET_SPARSE_BUFFER element is as follows:



SetSparse (1 byte): A Boolean (section 2.1.8) value.

A FALSE value will cause the file system to attempt to "unsparse" the file by allocating clusters for any regions of the file that are currently sparsed. If the entire file is successfully unsparsed, the sparse flag is cleared for the file. If an error is encountered during unsparsing, any regions of the file that were unsparsed MAY<50> remain unsparsed.

A TRUE value will cause the sparse flag for the file to set. Currently allocated clusters SHOULD NOT<51> be deallocated.

2.3.56 FSCTL_SET_SPARSE Reply

This message returns the results of the FSCTL_SET_SPARSE request.

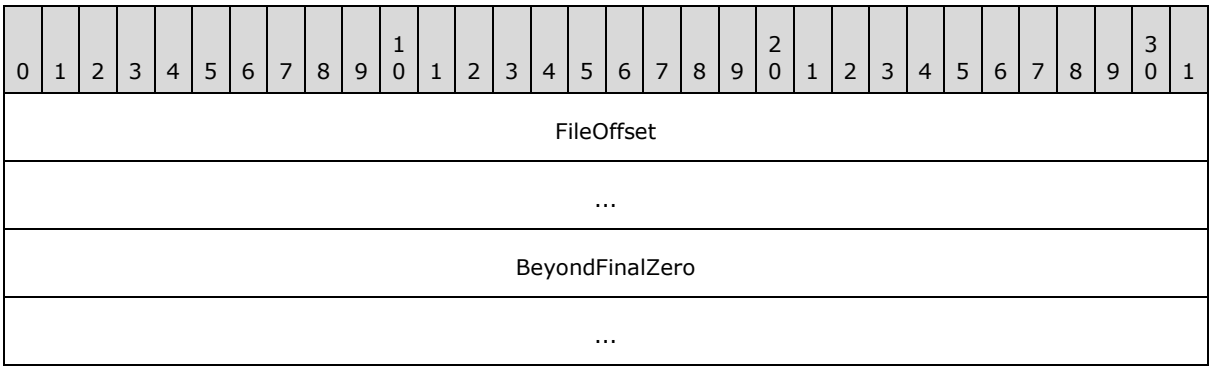
The only data item this message returns is a status code, as specified in [MS-ERREF] section 2.3. The status code returned directly by the function that processes this FSCTL MUST be STATUS_SUCCESS or one of the following.

Error code	Meaning
STATUS_INVALID_PARAMETER 0xC000000D	The handle is not to a file, or the input buffer length is nonzero and is less than the size of a FILE_SET_SPARSE_BUFFER structure.
STATUS_ACCESS_DENIED 0xC0000022	The handle is not open with write data or write attribute access.

2.3.57 FSCTL_SET_ZERO_DATA Request

The FSCTL_SET_ZERO_DATA request message requests that the server fill the specified range of the file (associated with the handle on which this FSCTL was invoked) with zeros. The message contains a FILE_ZERO_DATA_INFORMATION element.

The FILE_ZERO_DATA_INFORMATION element is as follows.



FileOffset (8 bytes): A 64-bit signed integer that contains the file offset of the start of the range to set to zeros, in bytes. The value of this field MUST be greater than or equal to 0.

BeyondFinalZero (8 bytes): A 64-bit signed integer that contains the byte offset of the first byte beyond the last zeroed byte. The value of this field MUST be greater than or equal to 0.

How an implementation zeros data within a file is implementation-dependent. A file system MAY choose to deallocate regions of disk space that have been zeroed.<52>

2.3.58 FSCTL_SET_ZERO_DATA Reply

This message returns the results of the FSCTL_SET_ZERO_DATA request.

The only data item this message returns is a status code, as specified in [MS-ERREF] section 2.3. The status code returned directly by the function that processes this FSCTL MUST be STATUS_SUCCESS or one of the following.

Error code	Meaning
STATUS_INVALID_PARAMETER 0xC000000D	The handle is not to a file, or input buffer length is not equal to the size of a FILE_ZERO_DATA_INFORMATION structure, or the given **FileOffset** is less than zero, or the given **BeyondFinalZero** is less than zero, or the given **FileOffset** is greater than the given BeyondFinalZero.
STATUS_ACCESS_DENIED 0xC0000022	The handle is not open with write data or write attribute access.

2.3.59 FSCTL_SET_ZERO_ON_DEALLOCATION Request

This message requests that the server fill the clusters of the target file with zeros when they are deallocated.<53> This is used to set a file to secure delete mode, which ensures that data will be zeroed upon file truncation or deletion.

There are several side effects associated with this operation.

- If the file is resident, it is converted to non-resident and the resident portion is zeroed.
- When reallocating ranges of a compressed file, the clusters are both zeroed and then replaced with a cluster representing compressed zeros before being reallocated.

This message does not contain any additional data elements.

2.3.60 FSCTL_SET_ZERO_ON_DEALLOCATION Reply

This message returns the results of the FSCTL_SET_ZERO_ON_DEALLOCATION request. The only data item this message returns is a status code, as specified in [MS-ERREF] section 2.3. The status code returned directly by the function that processes this FSCTL MUST be STATUS_SUCCESS or the following.

Error code	Meaning
STATUS_ACCESS_DENIED 0xC0000022	Zero on deallocation can only be set on a user file opened for write access and cannot be set on a directory.

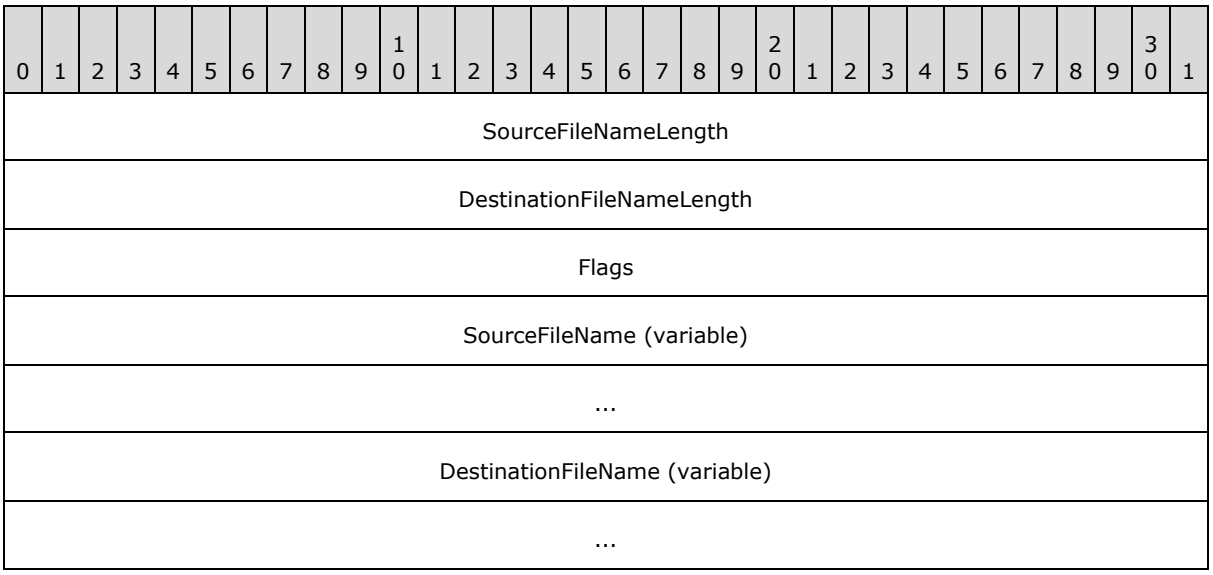
2.3.61 FSCTL_SIS_COPYFILE Request

The FSCTL_SIS_COPYFILE request message requests that the server use the single-instance storage (SIS) filter to copy a file. The message contains an SI_COPYFILE data element. For more information about single-instance storage, see [SIS].

If the SIS filter is installed on the server, it will attempt to copy the specified source file to the specified destination file by creating an SIS link instead of actually copying the file data. If necessary and allowed, the source file is placed under SIS control before the destination file is created.

This FSCTL can be issued against either a file or directory handle. The source and destination files MUST reside on the volume associated with the given handle.

The SI_COPYFILE data element is as follows.



SourceFileNameLength (4 bytes): A 32-bit unsigned integer that contains the size, in bytes, of the **SourceFileName** element, including a terminating-Unicode null character.

DestinationFileNameLength (4 bytes): A 32-bit unsigned integer that contains the size, in bytes, of the **DestinationFileName** element, including a terminating-Unicode null character.

Flags (4 bytes): A 32-bit unsigned integer that contains zero or more of the following flag values. Flag values not specified in the following table SHOULD be set to 0, and MUST be ignored.

Value	Meaning
COPYFILE_SIS_LINK 0x00000001	If this flag is set, only create the destination file if the source file is already under SIS control. If the source file is not under SIS control, the FSCTL returns STATUS_OBJECT_TYPE_MISMATCH. If this flag is not specified, place the source file under SIS control (if it is not already under SIS control), and create the destination file.
COPYFILE_SIS_REPLACE 0x00000002	If this flag is set, create the destination file if it does not exist; if it does exist, overwrite it. If this flag is not specified, create the destination file if it does not exist; if it does exist, the FSCTL returns STATUS_OBJECT_NAME_COLLISION.

SourceFileName (variable): A null-terminated Unicode string containing the source file name.

DestinationFileName (variable): A null-terminated Unicode string containing the destination file name.<54>

2.3.62 FSCTL_SIS_COPYFILE Reply

This message returns the results of the FSCTL_SIS_COPYFILE request.

The only data item this message returns is a status code, as specified in [MS-ERREF] section 2.3. The status code returned directly by the function that processes this FSCTL MUST be STATUS_SUCCESS or one of the following.

Error code	Meaning
STATUS_INVALID_PARAMETER 0xC000000D	The input buffer is NULL, or the input buffer length is less than the size of the SI_COPYFILE structure, or the given **SourceFileNameLength** or **DestinationFileNameLength** is less than 2 or greater than the buffer length, or the given **SourceFileNameLength** plus **DestinationFileNameLength** is greater than the length of the given **SourceFileName** plus **DestinationFileName** in the input buffer, or the given **SourceFileName** or **DestinationFileName** is NULL, or the given **SourceFileName** or **DestinationFileName** is not null-terminated.
STATUS_OBJECT_NAME_NOT_FOUND 0xC0000034	The source file does not exist.
STATUS_OBJECT_NAME_COLLISION 0xC0000035	The COPYFILE_SIS_REPLACE flag was not specified, and the destination file exists, or the source and destination file are the same.
STATUS_OBJECT_TYPE_MISMATCH 0xC0000024	The COPYFILE_SIS_LINK flag was specified, and the source file is not under SIS control.
STATUS_NOT_SAME_DEVICE 0xC00000D4	The source and destination file names are not located on the same volume, or the source and destination file names are located on the same volume, but it is not the volume associated with the handle on which the FSCTL was performed.
STATUS_INVALID_DEVICE_REQUEST 0xC0000010	The single-instance storage (SIS) filter is not installed on the server.
STATUS_FILE_IS_A_DIRECTORY 0xC00000BA	The source or destination file is a directory.
STATUS_ACCESS_DENIED 0xC0000022	The caller is not an administrator.

2.3.63 FSCTL_WRITE_USN_CLOSE_RECORD Request

This message requests that the server generate a record in the server's file system change journal stream for the file or directory associated with the handle on which this FSCTL was invoked, indicating that the file or directory was closed. This FSCTL can be called independently of the actual file close operation to write a USN record and cause a post of any pending USN updates for the indicated file.

No data structure is associated with this request.

2.3.64 FSCTL_WRITE_USN_CLOSE_RECORD Reply

This message returns the results of the FSCTL_WRITE_USN_CLOSE_RECORD request as a single field, **Usn**, which is a 64-bit signed integer that contains the server file system's USN (update sequence number) for the file or directory. This value MUST be greater than or equal to 0.

This message returns a status code, as specified in [MS-ERREF] section 2.3. The status code returned directly by the function that processes this FSCTL MUST be STATUS_SUCCESS or one of the following.

Error code	Meaning
STATUS_INVALID_PARAMETER 0xC000000D	The handle is not to a file or directory, or the length of the output buffer is less than the size of a 64-bit integer, or the output buffer does not begin on a 4-byte boundary.
STATUS_INVALID_DEVICE_REQUEST 0xC0000010	The file system does not support the use of a USN change journal.

2.4 File Information Classes

File information classes are numerical values (specified by the Level column in the following table) that specify what information for a file is to be queried or set. File information classes can require additional information to be included in the query or the response. When appropriate, the additional information is detailed in the file information class description. The table indicates which file information classes are supported for query and set operations.<55>

File information class	Level	Uses
FileAccessInformation	8	Query
FileAlignmentInformation	17	Query
FileAllInformation	18	Query
FileAllocationInformation	19	Set
FileAlternateNameInformation	21	Query
FileAttributeTagInformation	35	Query
FileBasicInformation	4	Query, Set
FileBothDirectoryInformation	3	Query
FileCompressionInformation	28	Query
FileDirectoryInformation	1	Query
FileDispositionInformation	13	Set
FileEaInformation	7	Query
FileEndOfFileInformation	20	Set
FileFullDirectoryInformation	2	Query
FileFullEaInformation	15	Query, Set
FileHardLinkInformation	46	LOCAL<56>
FileIdBothDirectoryInformation	37	Query
FileIdFullDirectoryInformation	38	Query

File information class	Level	Uses
FileIdGlobalTxDirectoryInformation	50	LOCAL<57>
FileInternalInformation	6	Query
FileLinkInformation	11	Set
FileMailslotQueryInformation	26	LOCAL<58>
FileMailslotSetInformation	27	LOCAL<59>
FileModeInformation	16	Query, Set<60>
FileMoveClusterInformation	31	<61>
FileNameInformation	9	LOCAL<62>
FileNamesInformation	12	Query
FileNetworkOpenInformation	34	Query
FileNormalizedNameInformation	48	<63>
FileObjectIdInformation	29	LOCAL<64>
FilePipeInformation	23	Query, Set
FilePipeLocalInformation	24	Query
FilePipeRemoteInformation	25	Query
FilePositionInformation	14	Query, Set
FileQuotaInformation	32	Query, Set<65>
FileRenameInformation	10	Set
FileReparsePointInformation	33	LOCAL<66>
FileSfioReserveInformation	44	LOCAL<67>
FileSfioVolumeInformation	45	<68>
FileShortNameInformation	40	Set
FileStandardInformation	5	Query
FileStandardLinkInformation	54	LOCAL<69>
FileStreamInformation	22	Query
FileTrackingInformation	36	LOCAL<70>
FileValidDataLengthInformation	39	Set

If a file system does not support a specific File Information Class, STATUS_INVALID_PARAMETER MUST be returned.

2.4.1 FileAccessInformation

This information class is used to query the access rights of a file that were granted when the file was opened.

A **FILE_ACCESS_INFORMATION** data element, defined as follows, is returned by the server.

0	1	2	3	4	5	6	7	8	9	10	1	2	3	4	5	6	7	8	9	20	1	2	3	4	5	6	7	8	9	30	1
AccessFlags																															

AccessFlags (4 bytes): A 32-bit unsigned integer that MUST contain values specified in [MS-SMB2] section 2.2.13.1.

This operation returns a status code, as specified in [MS-ERREF] section 2.3. The status code returned directly by the function that processes this file information class MUST be STATUS_SUCCESS or one of the following.

Error code	Meaning
STATUS_INFO_LENGTH_MISMATCH 0xC0000004	The specified information record length does not match the length that is required for the specified information class.

2.4.2 FileAllInformation

This information class is used to query a collection of file information structures.

A **FILE_ALL_INFORMATION** data element, defined as follows, is returned by the server.

0	1	2	3	4	5	6	7	8	9	10	1	2	3	4	5	6	7	8	9	20	1	2	3	4	5	6	7	8	9	30	1
BasicInformation																															
...																															
...																															
...																															
...																															
...																															
...																															
...																															
(BasicInformation cont'd for 2 rows)																															

StandardInformation
...
...
...
...
...
InternalInformation
...
EaInformation
AccessInformation
PositionInformation
...
ModeInformation
AlignmentInformation
NameInformation (variable)
...

BasicInformation (40 bytes): A FILE_BASIC_INFORMATION structure specified in section 2.4.7.

StandardInformation (24 bytes): A FILE_STANDARD_INFORMATION structure specified in section 2.4.38.

InternalInformation (8 bytes): A FILE_INTERNAL_INFORMATION structure specified in section 2.4.20.

EaInformation (4 bytes): A FILE_EA_INFORMATION structure specified in section 2.4.12.

AccessInformation (4 bytes): A FILE_ACCESS_INFORMATION structure specified in section 2.4.1.

PositionInformation (8 bytes): A FILE_POSITION_INFORMATION structure specified in section 2.4.32.

ModeInformation (4 bytes): A FILE_MODE_INFORMATION structure specified in section 2.4.24.

AlignmentInformation (4 bytes): A FILE_ALIGNMENT_INFORMATION structure specified in section 2.4.3.

NameInformation (variable): A FILE_NAME_INFORMATION structure specified in section 2.4.25.

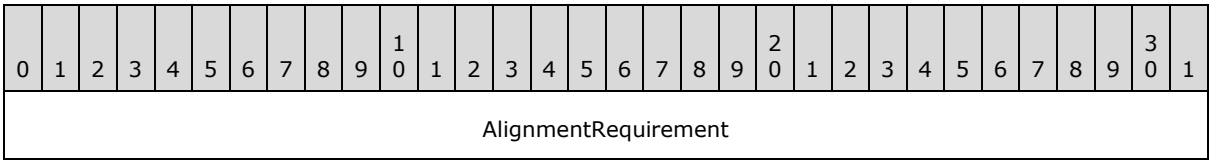
This operation returns a status code, as specified in [MS-ERREF] section 2.3. The status code returned directly by the function that processes this file information class MUST be STATUS_SUCCESS or one of the following.

Error code	Meaning
STATUS_INFO_LENGTH_MISMATCH 0xC0000004	The specified information record length does not match the length that is required for the specified information class.

2.4.3 FileAlignmentInformation

This information class is used to query the buffer alignment required by the underlying device.

A **FILE_ALIGNMENT_INFORMATION** data element, defined as follows, is returned by the server.



AlignmentRequirement (4 bytes): A 32-bit unsigned integer that MUST contain one of the following values.

Value	Meaning
FILE_BYTE_ALIGNMENT 0x00000000	If this value is specified, there are no alignment requirements for the device.
FILE_WORD_ALIGNMENT 0x00000001	If this value is specified, data MUST be aligned on a 2-byte boundary.
FILE_LONG_ALIGNMENT 0x00000003	If this value is specified, data MUST be aligned on a 4-byte boundary.
FILE_QUAD_ALIGNMENT 0x00000007	If this value is specified, data MUST be aligned on an 8-byte boundary.
FILE_OCTA_ALIGNMENT 0x0000000f	If this value is specified, data MUST be aligned on a 16-byte boundary.
FILE_32_BYTE_ALIGNMENT 0x0000001f	If this value is specified, data MUST be aligned on a 32-byte boundary.
FILE_64_BYTE_ALIGNMENT 0x0000003f	If this value is specified, data MUST be aligned on a 64-byte boundary.

Value	Meaning
FILE_128_BYTE_ALIGNMENT 0x0000007f	If this value is specified, data MUST be aligned on a 128-byte boundary.
FILE_256_BYTE_ALIGNMENT 0x000000ff	If this value is specified, data MUST be aligned on a 256-byte boundary.
FILE_512_BYTE_ALIGNMENT 0x000001ff	If this value is specified, data MUST be aligned on a 512-byte boundary.

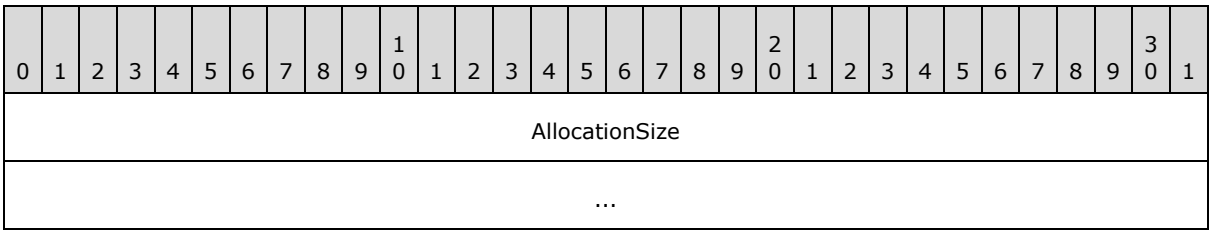
This operation returns a status code, as specified in [MS-ERREF] section 2.3. The status code returned directly by the function that processes this file information class MUST be STATUS_SUCCESS or one of the following.

Error code	Meaning
STATUS_INFO_LENGTH_MISMATCH 0xC0000004	The specified information record length does not match the length that is required for the specified information class.

2.4.4 FileAllocationInformation

This information class is used to set but not to query the allocation size for a file. The file system is passed a 64-bit signed integer containing the file allocation size, in bytes. The file system rounds the requested allocation size up to an integer multiple of the cluster size for nonresident files, or an implementation-defined multiple for resident files.<71><72> All unused allocation (beyond EOF) is freed on the last handle close.

A FILE_ALLOCATION_INFORMATION data element, defined as follows, is provided by the client.



AllocationSize (8 bytes): A 64-bit signed integer that contains the desired allocation to be used by the given file.

This operation returns a status code, as specified in [MS-ERREF] section 2.3. The status code returned directly by the function that processes this file information class MUST be STATUS_SUCCESS or one of the following.

Error code	Meaning
STATUS_INVALID_PARAMETER 0xC000000D	The handle is for a directory and not a file, or the allocation is greater than the maximum file size allowed.
STATUS_ACCESS_DENIED 0xC0000022	The handle was not opened to write file data or file attributes.
STATUS_DISK_FULL	The disk is full.

Error code	Meaning
0xC0000007	
STATUS_INFO_LENGTH_MISMATCH 0xC0000004	The specified information record length does not match the length that is required for the specified information class.

2.4.5 FileAlternateNameInformation

This information class is used to query **alternate name** information for a file. The alternate name for a file is its **8.3** format name (eight characters that appear before the "." and three characters that appear after). A file MAY have an alternate name to achieve compatibility with the 8.3 naming requirements of legacy applications. <73>

A FILE_NAME_INFORMATION (section 2.1.7) data element is returned by the server.

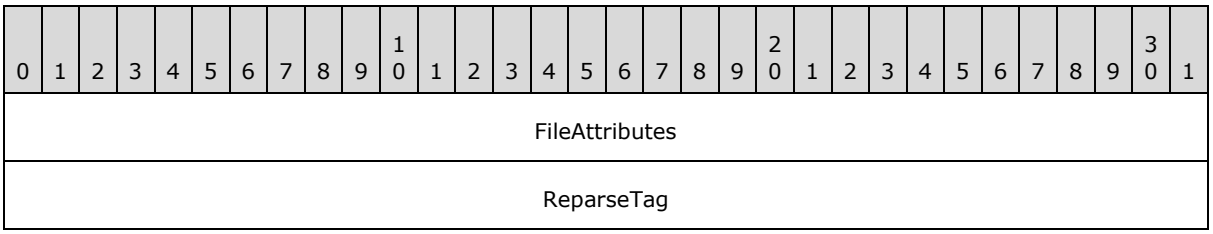
This operation returns a status code, as specified in [MS-ERREF] section 2.3. The status code returned directly by the function that processes this file information class MUST be STATUS_SUCCESS or one of the following.

Error code	Meaning
STATUS_INFO_LENGTH_MISMATCH 0xC0000004	The specified information record length does not match the length that is required for the specified information class.
STATUS_OBJECT_NAME_NOT_FOUND 0xC0000034	The object name is not found or is empty.
STATUS_BUFFER_OVERFLOW 0x80000005	The output buffer was filled before the complete name could be returned.

2.4.6 FileAttributeTagInformation

This information class is used to query for attribute and reparse tag information for a file.

A **FILE_ATTRIBUTE_TAG_INFORMATION** data element, defined as follows, is returned by the server.



FileAttributes (4 bytes): A 32-bit unsigned integer that contains the file attributes. Valid file attributes are as specified in section 2.6.

ReparseTag (4 bytes): A 32-bit unsigned integer that specifies the reparse point tag. If the **FileAttributes** member includes the FILE_ATTRIBUTE_REPARSE_POINT attribute flag, this member specifies the reparse tag. Otherwise, this member SHOULD be set to 0, and MUST be ignored. Section 2.1.2.1 contains more details on reparse tags.

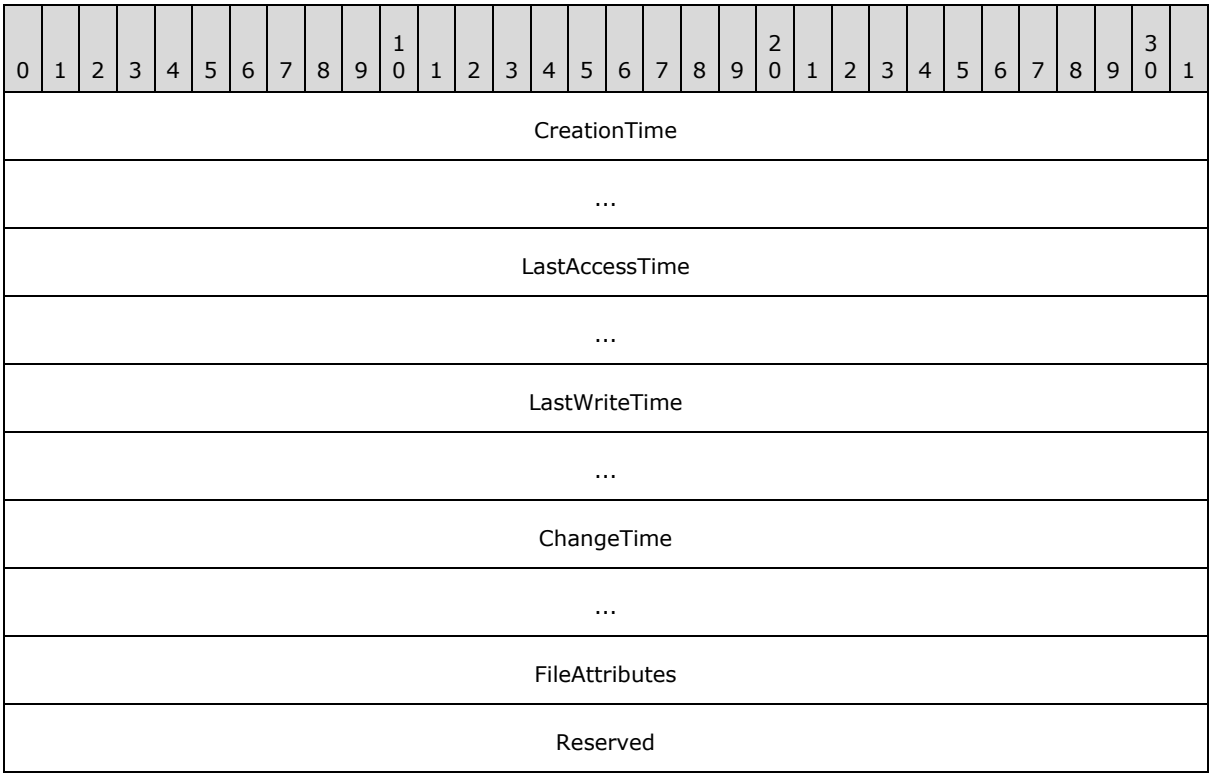
This operation returns a status code, as specified in [MS-ERREF] section 2.3. The status code returned directly by the function that processes this file information class MUST be STATUS_SUCCESS or one of the following.

Error code	Meaning
STATUS_INFO_LENGTH_MISMATCH 0xC0000004	The specified information record length does not match the length that is required for the specified information class.
STATUS_ACCESS_DENIED 0xC0000022	The handle was not opened to read file data or file attributes.

2.4.7 FileBasicInformation

This information class is used to query or set file information.

A FILE_BASIC_INFORMATION data element, defined as follows, is returned by the server or provided by the client.



CreationTime (8 bytes): The time when the file was created; see section 2.1.1. A valid time for this field is an integer greater than or equal to 0. When setting file attributes, a value of 0 indicates to the server that it MUST NOT change this attribute. When setting file attributes, a value of -1 indicates to the server that it MUST NOT change this attribute for all subsequent operations on the same file handle. This field MUST NOT be set to a value less than -1.

LastAccessTime (8 bytes): The last time the file was accessed; see section 2.1.1. A valid time for this field is an integer greater than or equal to 0. When setting file attributes, a value of 0

indicates to the server that it MUST NOT change this attribute. When setting file attributes, a value of -1 indicates to the server that it MUST NOT change this attribute for all subsequent operations on the same file handle. This field MUST NOT be set to a value less than -1. <74>

LastWriteTime (8 bytes): The last time information was written to the file; see section 2.1.1. A valid time for this field is an integer greater than or equal to 0. When setting file attributes, a value of 0 indicates to the server that it MUST NOT change this attribute. When setting file attributes, a value of -1 indicates to the server that it MUST NOT change this attribute for all subsequent operations on the same file handle. This field MUST NOT be set to a value less than -1. <75>

ChangeTime (8 bytes): The last time the file was changed; see section 2.1.1. A valid time for this field is an integer greater than or equal to 0. When setting file attributes, a value of 0 indicates to the server that it MUST NOT change this attribute. When setting file attributes, a value of -1 indicates to the server that it MUST NOT change this attribute for all subsequent operations on the same file handle. This field MUST NOT be set to a value less than -1. <76>

FileAttributes (4 bytes): A 32-bit unsigned integer that contains the file attributes. Valid file attributes are specified in section 2.6.

Reserved (4 bytes): A 32-bit field. This field is reserved. This field can be set to any value, and MUST be ignored.

This operation returns a status code, as specified in [MS-ERREF] section 2.3. The status code returned directly by the function that processes this file information class MUST be STATUS_SUCCESS or one of the following.

Error code	Meaning
STATUS_INFO_LENGTH_MISMATCH 0xC0000004	The specified information record length does not match the length that is required for the specified information class.
STATUS_ACCESS_DENIED 0xC0000022	The handle was not opened to read file data or file attributes.

2.4.8 FileBothDirectoryInformation

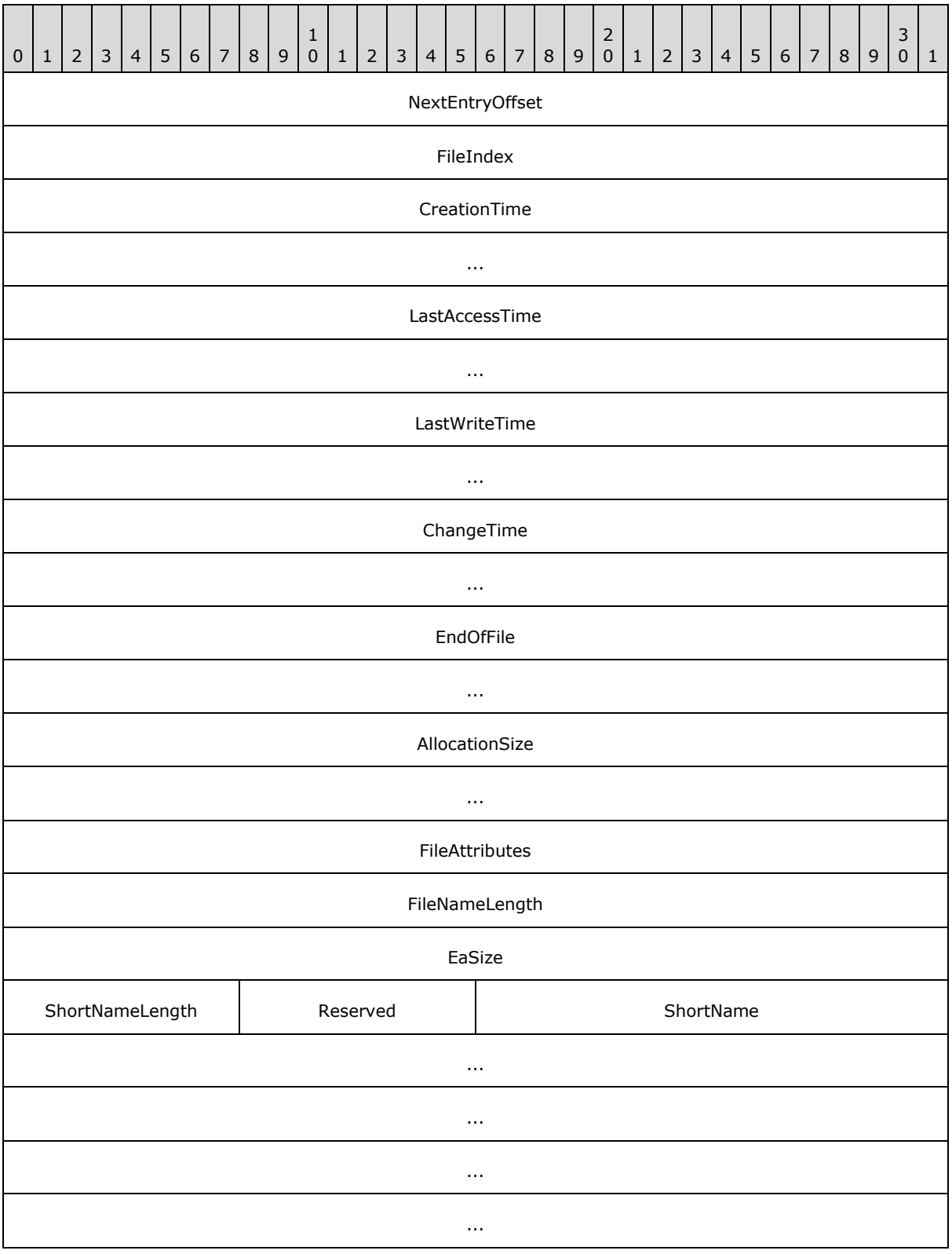
This information class is used in directory enumeration to return detailed information about the contents of a directory.

This information class returns a list that contains a **FILE_BOTH_DIR_INFORMATION** data element for each file or directory within the target directory. This list MUST reflect the presence of a subdirectory named "." (synonymous with the target directory itself) within the target directory and one named ".." (synonymous with the parent directory of the target directory). For more details, see section 2.1.5.1.

This information class differs from FileDirectoryInformation (section 2.4.10) in that it includes short names in the returns list.

When multiple **FILE_BOTH_DIR_INFORMATION** data elements are present in the buffer, each MUST be aligned on an 8-byte boundary. Any bytes inserted for alignment SHOULD be set to zero, and the receiver MUST ignore them. No padding is required following the last data element.

A **FILE_BOTH_DIR_INFORMATION** data element is as follows.



...	
...	FileName (variable)
...	

NextEntryOffset (4 bytes): A 32-bit unsigned integer that contains the byte offset from the beginning of this entry, at which the next FILE_BOTH_DIR_INFORMATION entry is located, if multiple entries are present in a buffer. This member is zero if no other entries follow this one. An implementation MUST use this value to determine the location of the next entry (if multiple entries are present in a buffer).

FileIndex (4 bytes): A 32-bit unsigned integer that contains the byte offset of the file within the parent directory. For file systems in which the position of a file within the parent directory is not fixed and can be changed at any time to maintain sort order, this field SHOULD be set to 0, and MUST be ignored.<77>

CreationTime (8 bytes): The time when the file was created; see section 2.1.1. This value MUST be greater than or equal to 0.

LastAccessTime (8 bytes): The last time the file was accessed; see section 2.1.1. This value MUST be greater than or equal to 0.

LastWriteTime (8 bytes): The last time information was written to the file; see section 2.1.1. This value MUST be greater than or equal to 0.

ChangeTime (8 bytes): The last time the file was changed; see section 2.1.1. This value MUST be greater than or equal to 0.

EndOfFile (8 bytes): A 64-bit signed integer that contains the absolute new end-of-file position as a byte offset from the start of the file. EndOfFile specifies the offset to the byte immediately following the last valid byte in the file. Because this value is zero-based, it actually refers to the first free byte in the file. That is, it is the offset from the beginning of the file at which new bytes appended to the file will be written. The value of this field MUST be greater than or equal to 0.

AllocationSize (8 bytes): A 64-bit signed integer that contains the file allocation size, in bytes. The value of this field MUST be an integer multiple of the cluster size.

FileAttributes (4 bytes): A 32-bit unsigned integer that contains the file attributes. Valid file attributes are specified in section 2.6.

FileNameLength (4 bytes): A 32-bit unsigned integer that specifies the length, in bytes, of the file name contained within the **FileName** member.

EaSize (4 bytes): A 32-bit unsigned integer that contains the combined length, in bytes, of the extended attributes (EA) for the file.

ShortNameLength (1 byte): An 8-bit signed integer that specifies the length, in bytes, of the file name contained in the **ShortName** member. This value MUST be greater than or equal to 0.

Reserved (1 byte): Reserved for alignment. This field can contain any value and MUST be ignored.

ShortName (24 bytes): A sequence of Unicode characters containing the short (8.3) file name. When working with this field, use **ShortNameLength** to determine the length of the file name rather than assuming the presence of a trailing null delimiter.

FileName (variable): A sequence of Unicode characters containing the file name. When working with this field, use **FileNameLength** to determine the length of the file name rather than assuming the presence of a trailing null delimiter. Dot directory names are valid for this field. For more details, see section 2.1.5.1.

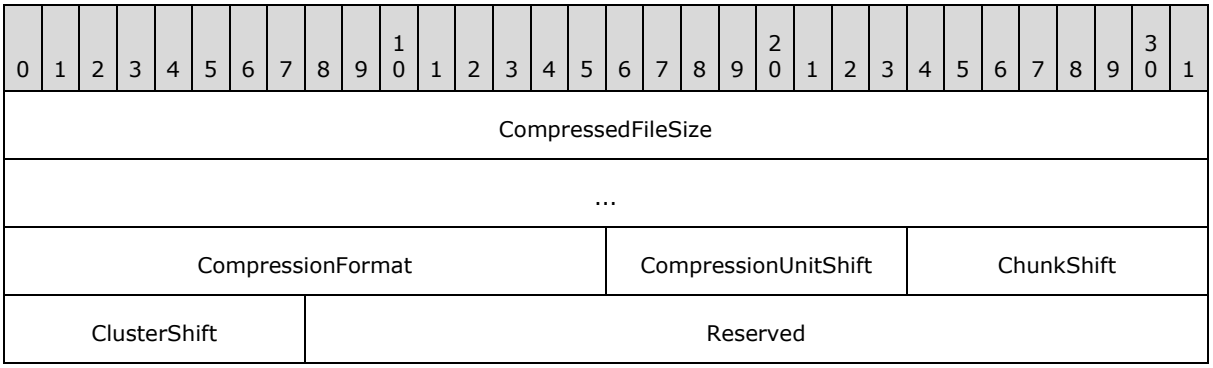
This operation returns a status code, as specified in [MS-ERREF] section 2.3. The status code returned directly by the function that processes this file information class MUST be STATUS_SUCCESS or one of the following.

Error code	Meaning
STATUS_INFO_LENGTH_MISMATCH 0xC0000004	The specified information record length does not match the length that is required for the specified information class.

2.4.9 FileCompressionInformation

This information class is used to query compression information for a file.

A FILE_COMPRESSION_INFORMATION data element, defined as follows, is returned by the server.



CompressedFileSize (8 bytes): A 64-bit signed integer that contains the size, in bytes, of the compressed file. This value MUST be greater than or equal to 0.

CompressionFormat (2 bytes): A 16-bit unsigned integer that contains the compression format. The actual compression operation associated with each of these compression format values is implementation-dependent. An implementation can link any local compression algorithm with the values described in the following table because the compressed data does not travel across the wire in the context of FSCTL, FileInformation class, or FileSystemInformation class requests or replies.<78>

Value	Meaning
COMPRESSION_FORMAT_NONE 0x0000	The file or directory is not compressed.
COMPRESSION_FORMAT_LZNT1 0x0002	The file or directory is compressed by using the LZNT1 compression algorithm.

Value	Meaning
All other values	Reserved for future use.

CompressionUnitShift (1 byte): An 8-bit unsigned integer that contains the **compression unit shift**, which is the number of bits by which to left-shift a 1 bit to arrive at the compression unit size. The compression unit size is the number of bytes in a compression unit, that is, the number of bytes to be compressed. This value is implementation-defined.<79>

ChunkShift (1 byte): An 8-bit unsigned integer that contains the compression **chunk** size shift, which is the number of bits by which to left-shift a 1 bit to arrive at the compression chunk size. The chunk size is the number of bytes that the operating system's implementation of the Lempel-Ziv compression algorithm tries to compress at one time. This value is implementation-defined.<80>

ClusterShift (1 byte): An 8-bit unsigned integer that contains the cluster size shift, which is the number of bits by which to left-shift a 1 bit to arrive at the cluster size. The cluster size specifies the amount of space that must be saved by compression to successfully compress a compression unit. If a cluster size amount of space is not saved by compression, the data in that compression unit is stored uncompressed. Each successfully compressed compression unit MUST occupy at least one cluster less than the uncompressed compression unit. This value is implementation-defined.<81>

Reserved (3 bytes): A 24-bit reserved value. This field SHOULD be set to 0, and MUST be ignored.

This operation returns a status code, as specified in [MS-ERREF] section 2.3. The status code returned directly by the function that processes this file information class MUST be STATUS_SUCCESS or one of the following.

Error code	Meaning
STATUS_INFO_LENGTH_MISMATCH 0xC0000004	The specified information record length does not match the length that is required for the specified information class.
STATUS_BUFFER_OVERFLOW 0x80000005	The data was too large to fit into the specified buffer. No data is returned.

2.4.10 FileDirectoryInformation

This information class is used in directory enumeration to return detailed information about the contents of a directory.

This information class returns a list that contains a **FILE_DIRECTORY_INFORMATION** data element for each file or directory within the target directory. This list MUST reflect the presence of a subdirectory named "." (synonymous with the target directory itself) within the target directory and one named ".." (synonymous with the parent directory of the target directory). For more details, see section 2.1.5.1.

When multiple **FILE_DIRECTORY_INFORMATION** data elements are present in the buffer, each MUST be aligned on an 8-byte boundary. Any bytes inserted for alignment SHOULD be set to zero, and the receiver MUST ignore them. No padding is required following the last data element.

A **FILE_DIRECTORY_INFORMATION** data element is as follows.

0	1	2	3	4	5	6	7	8	9	10	1	2	3	4	5	6	7	8	9	20	1	2	3	4	5	6	7	8	9	30	1
NextEntryOffset																															
FileIndex																															
CreationTime																															
...																															
LastAccessTime																															
...																															
LastWriteTime																															
...																															
ChangeTime																															
...																															
EndOfFile																															
...																															
AllocationSize																															
...																															
FileAttributes																															
FileNameLength																															
FileName (variable)																															
...																															

NextEntryOffset (4 bytes): A 32-bit unsigned integer that contains the byte offset from the beginning of this entry, at which the next FILE_DIRECTORY_INFORMATION entry is located, if multiple entries are present in a buffer. This member MUST be zero if no other entries follow this one. An implementation MUST use this value to determine the location of the next entry (if multiple entries are present in a buffer).

FileIndex (4 bytes): A 32-bit unsigned integer that contains the byte offset of the file within the parent directory. For file systems in which the position of a file within the parent directory

is not fixed and can be changed at any time to maintain sort order, this field SHOULD be set to 0 and MUST be ignored.<82>

CreationTime (8 bytes): The time when the file was created; see section 2.1.1. This value MUST be greater than or equal to 0.

LastAccessTime (8 bytes): The last time the file was accessed; see section 2.1.1. This value MUST be greater than or equal to 0.

LastWriteTime (8 bytes): The last time information was written to the file; see section 2.1.1. This value MUST be greater than or equal to 0.

ChangeTime (8 bytes): The last time the file was changed; see section 2.1.1. This value MUST be greater than or equal to 0.

EndOfFile (8 bytes): A 64-bit signed integer that contains the absolute new end-of-file position as a byte offset from the start of the file. EndOfFile specifies the offset to the byte immediately following the last valid byte in the file. Because this value is zero-based, it actually refers to the first free byte in the file. That is, it is the offset from the beginning of the file at which new bytes appended to the file will be written. The value of this field MUST be greater than or equal to 0.

AllocationSize (8 bytes): A 64-bit signed integer that contains the file allocation size, in bytes. The value of this field MUST be an integer multiple of the cluster size.

FileAttributes (4 bytes): A 32-bit unsigned integer that contains the file attributes. Valid attributes are as specified in section 2.6.

FileNameLength (4 bytes): A 32-bit unsigned integer that specifies the length, in bytes, of the file name contained within the **FileName** member.

FileName (variable): A sequence of Unicode characters containing the file name. When working with this field, use **FileNameLength** to determine the length of the file name rather than assuming the presence of a trailing null delimiter. Dot directory names are valid for this field. For more details, see section 2.1.5.1.

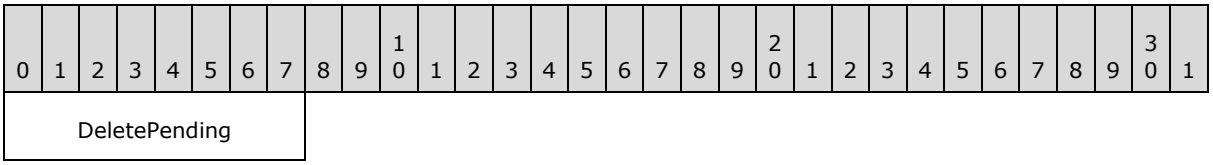
This operation returns a status code, as specified in [MS-ERREF] section 2.3. The status code returned directly by the function that processes this file information class MUST be STATUS_SUCCESS or one of the following.

Error code	Meaning
STATUS_INFO_LENGTH_MISMATCH 0xC0000004	The specified information record length does not match the length that is required for the specified information class.

2.4.11 FileDispositionInformation

This information class is used to mark a file for deletion.

A FILE_DISPOSITION_INFORMATION data element, defined as follows, is provided by the client.



DeletePending (1 byte): An 8-bit field that is set to 1 to indicate that a file SHOULD be deleted when it is closed; otherwise, 0.<83>

For a discussion of file deletion semantics, see [FSBO].

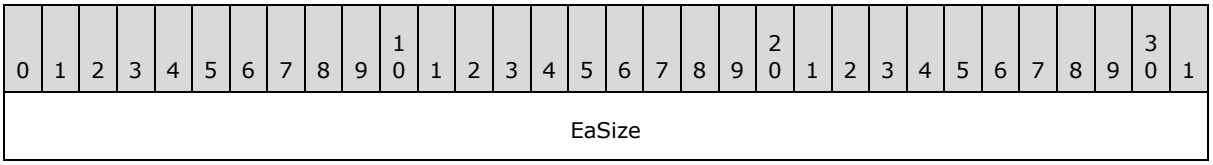
This operation returns a status code, as specified in [MS-ERREF] section 2.3. The status code returned directly by the function that processes this file information class MUST be STATUS_SUCCESS or one of the following.

Error code	Meaning
STATUS_ACCESS_DENIED 0xC0000022	The handle was not opened with delete access.
STATUS_INFO_LENGTH_MISMATCH 0xC0000004	The specified information record length does not match the length that is required for the specified information class.

2.4.12 FileEaInformation

This information class is used to query for the size of the extended attributes (EA) for a file. An extended attribute is a piece of application-specific metadata that an application can link with a file that is not part of the file's data. For more information about extended attributes, see [MS-CIFS] section 2.2.1.2.

A FILE_EA_INFORMATION data element, defined as follows, is returned by the server.



EaSize (4 bytes): A 32-bit unsigned integer that contains the combined length, in bytes, of the extended attributes (EA) for the file.

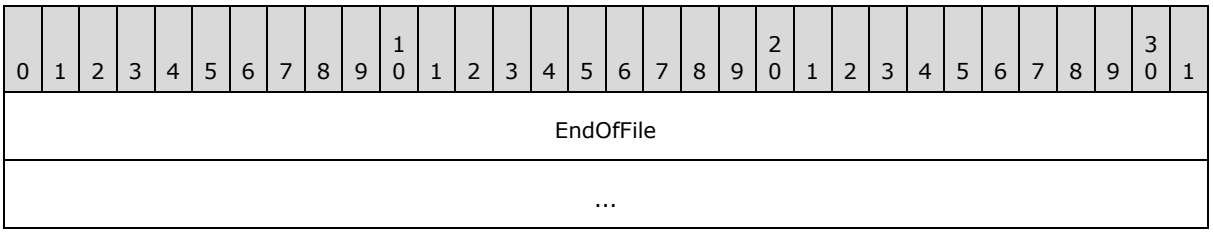
This operation returns a status code, as specified in [MS-ERREF] section 2.3. The status code returned directly by the function that processes this file information class MUST be STATUS_SUCCESS or one of the following.

Error code	Meaning
STATUS_INFO_LENGTH_MISMATCH 0xC0000004	The specified information record length does not match the length that is required for the specified information class.

2.4.13 FileEndOfFileInformation

This information class is used to set end-of-file information for a file.

A FILE_END_OF_FILE_INFORMATION data element, defined as follows, is provided by the client.



EndOfFile (8 bytes): A 64-bit signed integer that contains the absolute new end of file position as a byte offset from the start of the file. EndOfFile specifies the offset from the beginning of the file of the byte following the last byte in the file. That is, it is the offset from the beginning of the file at which new bytes appended to the file will be written. The value of this field MUST be greater than or equal to 0.

This operation returns a status code, as specified in [MS-ERREF] section 2.3. The status code returned directly by the function that processes this file information class MUST be STATUS_SUCCESS or one of the following.

Error code	Meaning
STATUS_INVALID_PARAMETER 0xC000000D	The handle was for a directory and not a file, or the allocation is greater than the maximum file size allowed.
STATUS_ACCESS_DENIED 0xC0000022	The handle was not opened to read file data or file attributes.
STATUS_DISK_FULL 0xC0000007	The disk is full.
STATUS_INFO_LENGTH_MISMATCH 0xC0000004	The specified information record length does not match the length that is required for the specified information class.

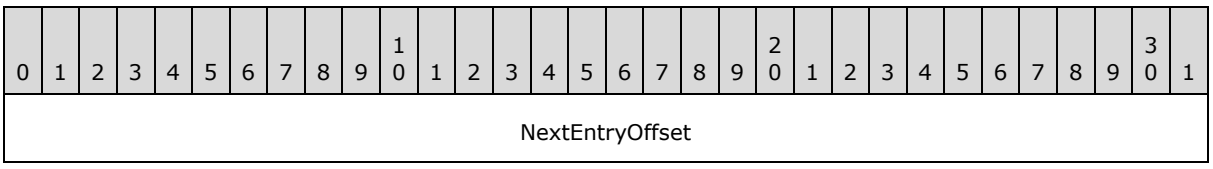
2.4.14 FileFullDirectoryInformation

This information class is used in directory enumeration to return detailed information about the contents of a directory.

This information class returns a list that contains a **FILE_FULL_DIR_INFORMATION** data element for each file or directory within the target directory. This list MUST reflect the presence of a subdirectory named "." (synonymous with the target directory itself) within the target directory and one named ".." (synonymous with the parent directory of the target directory). For more details, see section 2.1.5.1.

When multiple **FILE_FULL_DIR_INFORMATION** data elements are present in the buffer, each MUST be aligned on an 8-byte boundary; any bytes inserted for alignment SHOULD be set to zero, and the receiver MUST ignore them. No padding is required following the last data element.

A **FILE_FULL_DIR_INFORMATION** data element is as follows.



FileIndex
CreationTime
...
LastAccessTime
...
LastWriteTime
...
ChangeTime
...
EndOfFile
...
AllocationSize
...
FileAttributes
FileNameLength
EaSize
FileName (variable)
...

NextEntryOffset (4 bytes): A 32-bit unsigned integer that contains the byte offset from the beginning of this entry, at which the next FILE_FULL_DIR_INFORMATION entry is located, if multiple entries are present in a buffer. This member is zero if no other entries follow this one. An implementation MUST use this value to determine the location of the next entry (if multiple entries are present in a buffer).

FileIndex (4 bytes): A 32-bit unsigned integer that contains the byte offset of the file within the parent directory. For file systems such as NTFS, in which the position of a file within the parent directory is not fixed and can be changed at any time to maintain sort order, this field SHOULD be set to 0, and MUST be ignored.<84>

CreationTime (8 bytes): The time when the file was created; see section 2.1.1. This value MUST be greater than or equal to 0.

LastAccessTime (8 bytes): The last time the file was accessed; see section 2.1.1. This value MUST be greater than or equal to 0.

LastWriteTime (8 bytes): The last time information was written to the file; see section 2.1.1. This value MUST be greater than or equal to 0.

ChangeTime (8 bytes): The last time the file was changed; see section 2.1.1. This value MUST be greater than or equal to 0.

EndOfFile (8 bytes): A 64-bit signed integer that contains the absolute new end-of-file position as a byte offset from the start of the file. EndOfFile specifies the offset to the byte immediately following the last valid byte in the file. Because this value is zero-based, it actually refers to the first free byte in the file. That is, it is the offset from the beginning of the file at which new bytes appended to the file will be written. The value of this field MUST be greater than or equal to 0.

AllocationSize (8 bytes): A 64-bit signed integer that contains the file allocation size, in bytes. The value of this field MUST be an integer multiple of the cluster size.

FileAttributes (4 bytes): A 32-bit unsigned integer that contains the file attributes. For a list of valid file attributes, see section 2.6.

FileNameLength (4 bytes): A 32-bit unsigned integer that specifies the length, in bytes, of the file name contained within the **FileName** member.

EaSize (4 bytes): A 32-bit unsigned integer that contains the combined length, in bytes, of the extended attributes (EA) for the file.

FileName (variable): A sequence of Unicode characters containing the file name. When working with this field, use **FileNameLength** to determine the length of the file name rather than assuming the presence of a trailing null delimiter. Dot directory names are valid for this field. For more details, see section 2.1.5.1.

This operation returns a status code, as specified in [MS-ERREF] section 2.3. The status code returned directly by the function that processes this file information class MUST be STATUS_SUCCESS or one of the following.

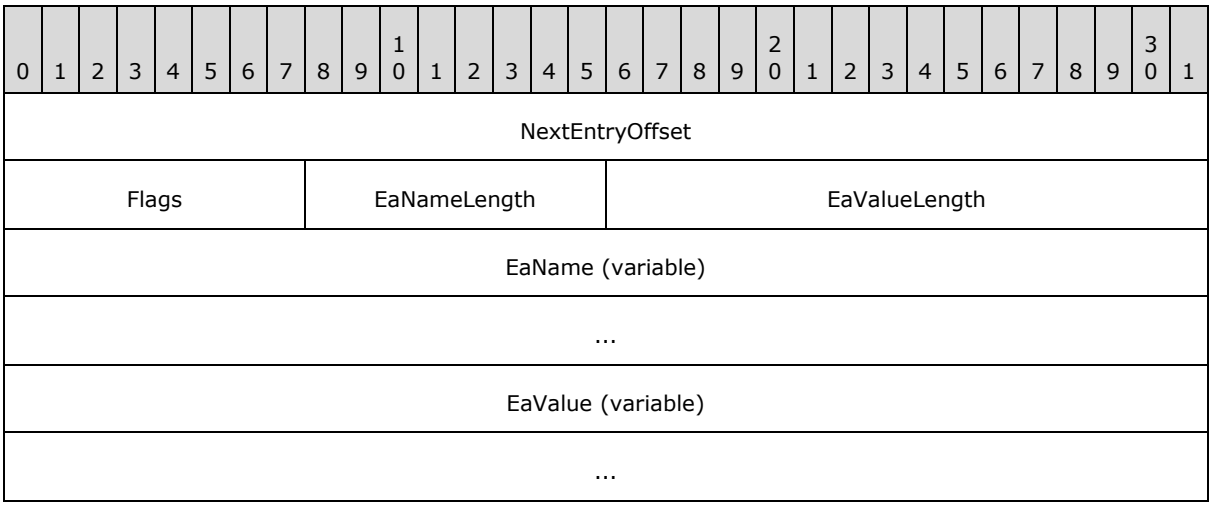
Error code	Meaning
STATUS_INFO_LENGTH_MISMATCH 0xC0000004	The specified information record length does not match the length that is required for the specified information class.

2.4.15 FileFullEaInformation

This information class is used to query or set extended attribute (EA) information for a file. For queries, the client provides a list of FILE_GET_EA_INFORMATION (section 2.4.15.1) structures, and a list of **FILE_FULL_EA_INFORMATION** structures is returned by the server. For setting EA information, the client provides a list of **FILE_FULL_EA_INFORMATION** structures, and a status code is returned by the server, as specified in [MS-ERREF] section 2.3.

When multiple **FILE_FULL_EA_INFORMATION** data elements are present in the buffer, each MUST be aligned on an 8-byte boundary. Any bytes inserted for alignment SHOULD be set to zero, and the receiver MUST ignore them. No padding is required following the last data element.

A **FILE_FULL_EA_INFORMATION** data element is as follows.



NextEntryOffset (4 bytes): A 32-bit unsigned 4-byte aligned integer that contains the byte offset from the beginning of this entry, at which the next FILE_FULL_EA_INFORMATION entry is located, if multiple entries are present in the buffer. This member MUST be zero if no other entries follow this one. An implementation MUST use this value to determine the location of the next entry (if multiple entries are present in a buffer).

Flags (1 byte): An 8-bit unsigned integer that MUST contain one of the following flag values.

Value	Meaning
0x00000000	If no flags are set, this EA does not prevent the file to which the EA belongs from being interpreted by applications that do not understand EAs.
FILE_NEED_EA 0x00000080	If this flag is set, the file to which the EA belongs cannot be interpreted by applications that do not understand EAs.

EaNameLength (1 byte): An 8-bit unsigned integer that contains the length, in bytes, of the extended attribute name in the **EaName** field. This value MUST NOT include the terminating null character to **EaName**.

EaValueLength (2 bytes): A 16-bit unsigned integer that contains the length, in bytes, of the extended attribute value in the **EaValue** field. When setting EA information, if this field is zero, then the given EaName and its current value are deleted from the given file.

EaName (variable): An array of 8-bit ASCII characters that contains the extended attribute name followed by a single terminating null character byte. The **EaName** MUST be less than 255 characters and MUST NOT contain any of the following characters:

ASCII values 0x00 - 0x1F, \ / : * ? " < > | , + = [ ] ;

EaValue (variable): An array of bytes that contains the extended attribute value. The length of this array is specified by the **EaValueLength** field.

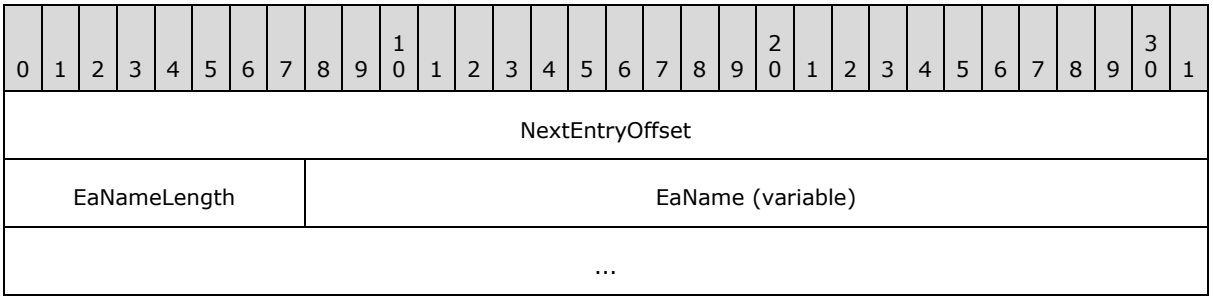
This operation returns a status code, as specified in [MS-ERREF] section 2.3. The status code returned directly by the function that processes this file information class MUST be STATUS_SUCCESS or one of the following.

Error code	Meaning
STATUS_ACCESS_DENIED 0xC0000022	The handle was not opened to read file data or file attributes.
STATUS_NO_EAS_ON_FILE 0xC0000052	The file for which EAs were requested has no EAs.
STATUS_BUFFER_TOO_SMALL 0xC0000023	The buffer is too small to contain the entry. No information has been written to the buffer.
STATUS_BUFFER_OVERFLOW 0x80000005	The output buffer was filled before all of the EA data could be returned. Only complete FILE_FULL_EA_INFORMATION structures are returned.
STATUS_INVALID_EA_NAME 0x80000013	The **Flags** field contains a value other than zero or FILE_NEED_EA, or the **EaName** field is longer than 255 characters, or it contains any of the following characters: ASCII values 0x00 - 0x1F, \ / : * ? " < > | , + = [ ] ;

2.4.15.1 FILE_GET_EA_INFORMATION

This data structure can be used to specify an explicit list of attributes to query via the FileFullEaInformation (section 2.4.15) information class. If no FILE_GET_EA_INFORMATION elements are specified, all extended attributes for the given file are returned.

When multiple FILE_GET_EA_INFORMATION data elements are present in the buffer, each MUST be aligned on a 4-byte boundary. Any bytes inserted for alignment SHOULD be set to zero, and the receiver MUST ignore them. No padding is required following the last data element.



NextEntryOffset (4 bytes): A 32-bit unsigned integer that contains the byte offset from the beginning of this entry, at which the next **FILE_GET_EA_INFORMATION** entry is located, if multiple entries are present in a buffer. This member MUST be zero if no other entries follow this one. An implementation MUST use this value to determine the location of the next entry (if multiple entries are present in a buffer).

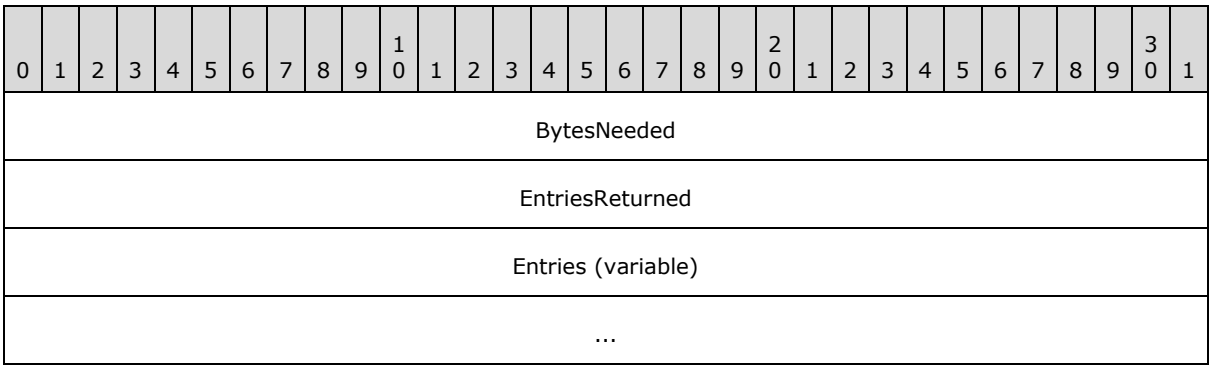
EaNameLength (1 byte): An 8-bit unsigned integer that contains the length, in bytes, of the **EaName** field. This value MUST NOT include the terminating null character to **EaName**.

EaName (variable): An array of 8-bit ASCII characters that contains the extended attribute name followed by a single terminating null character byte.

2.4.16 FileHardLinkInformation

This information class is used locally to query hard links to an existing file.<85> At least one name MUST be returned.

A **FILE_LINKS_INFORMATION** data element, defined as follows, is returned to the caller.



BytesNeeded (4 bytes): A 32-bit unsigned integer that MUST contain the number of bytes needed to hold all available names. This field MUST NOT be 0.

EntriesReturned (4 bytes): A 32-bit unsigned integer that MUST contain the number of FILE_LINK_ENTRY_INFORMATION structures that have been returned in the **Entries** field.

This field MUST return as many entries as will fit in available memory. A value of 0 indicates that there is not enough available memory to return any entry. The error STATUS_BUFFER_OVERFLOW (0x80000005) indicates that not all available entries were returned.

Entries (variable): A buffer that MUST contain the returned FILE_LINK_ENTRY_INFORMATION structures. It MUST be **BytesNeeded** bytes in size to return all of the available entries.

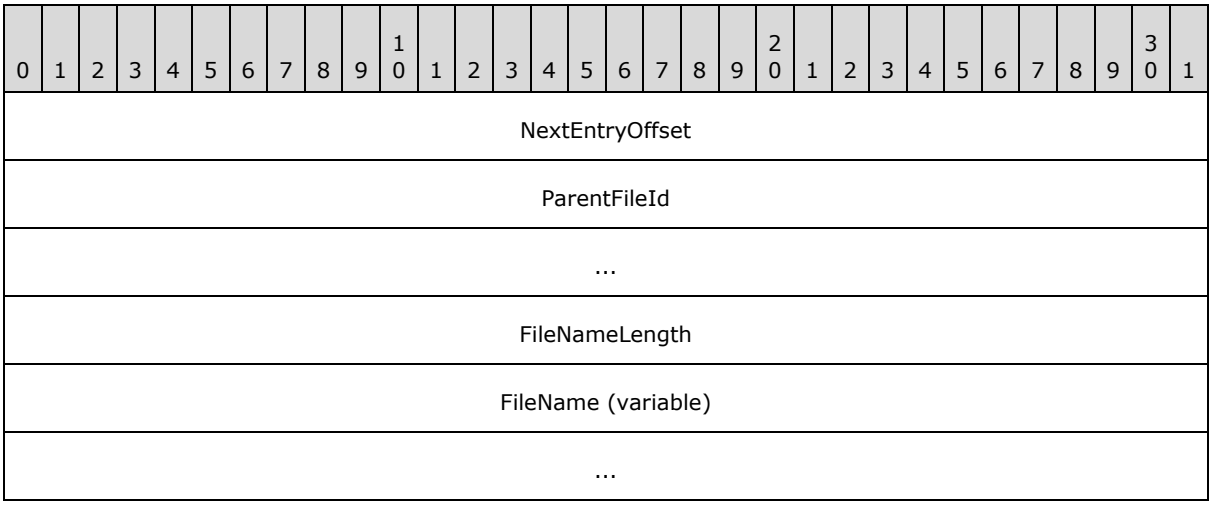
This operation returns a status code, as specified in [MS-ERREF] section 2.3. The status code returned directly by the function that processes this file information class MUST be STATUS_SUCCESS or one of the following.

Error code	Meaning
STATUS_NOT_SUPPORTED 0xC00000BB	The request is not supported.
STATUS_BUFFER_OVERFLOW 0x80000005	The output buffer was filled before all of the link information could be returned. Only complete FILE_LINK_ENTRY_INFORMATION structures are returned.
STATUS_INFO_LENGTH_MISMATCH 0xC0000004	The specified information record length does not match the length that is required for the specified information class.

2.4.16.1 FILE_LINK_ENTRY_INFORMATION

The FILE_LINK_ENTRY_INFORMATION packet is used to describe a single hard link to an existing file.

When multiple FILE_LINK_ENTRY_INFORMATION data elements are present in the buffer, each MUST be aligned on an 8-byte boundary. Any bytes inserted for alignment SHOULD be set to zero, and the receiver MUST ignore them. No padding is required following the last data element.



NextEntryOffset (4 bytes): A 32-bit unsigned integer that MUST specify the offset, in bytes, from the current FILE_LINK_ENTRY_INFORMATION structure to the next FILE_LINK_ENTRY_INFORMATION structure. A value of 0 indicates this is the last entry structure.

ParentFileId (8 bytes): A 64-bit signed integer that MUST contain the FileID of the parent directory of the given link.

FileNameLength (4 bytes): A 32-bit unsigned integer that MUST specify the length, in characters, of the FileName for the given link.

FileName (variable): A **WCHAR** array whose size is given by FileNameLength that MUST contain the Unicode string name of the given link.

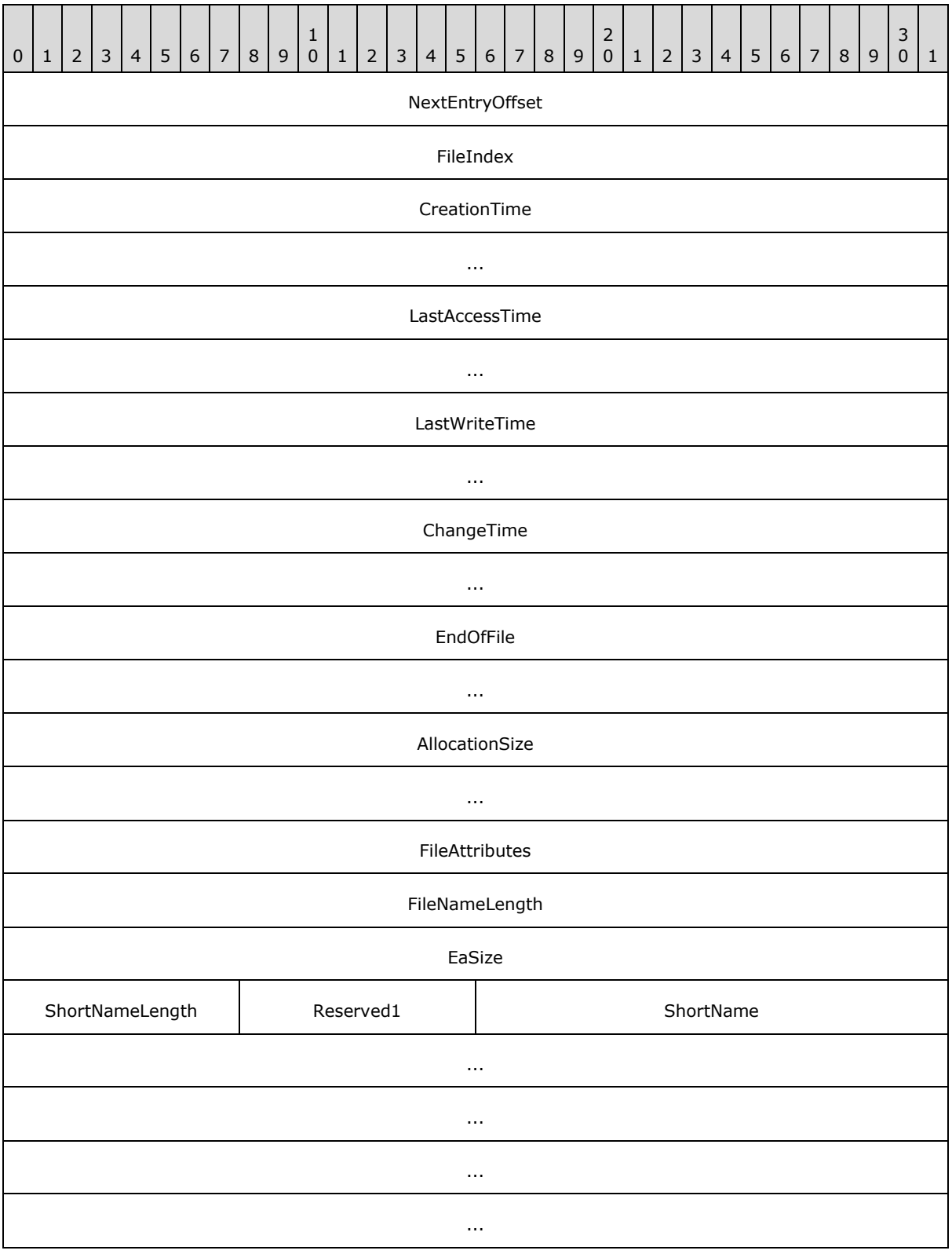
2.4.17 FileIdBothDirectoryInformation

This information class is used in directory enumeration to return detailed information about the contents of a directory.

This information class returns a list that contains a **FILE_ID_BOTH_DIR_INFORMATION** data element for each file or directory within the target directory. This list MUST reflect the presence of a subdirectory named "." (synonymous with the target directory itself) within the target directory and one named ".." (synonymous with the parent directory of the target directory). For more details, see section 2.1.5.1.

When multiple **FILE_ID_BOTH_DIR_INFORMATION** data elements are present in the buffer, each MUST be aligned on an 8-byte boundary. Any bytes inserted for alignment SHOULD be set to zero, and the receiver MUST ignore them. No padding is required following the last data element.

A **FILE_ID_BOTH_DIR_INFORMATION** data element is as follows.



...	
...	Reserved2
FileId	
...	
FileName (variable)	
...	

NextEntryOffset (4 bytes): A 32-bit unsigned integer that contains the byte offset from the beginning of this entry, at which the next FILE_ID_BOTH_DIR_INFORMATION entry is located, if multiple entries are present in the buffer. This member MUST be zero if no other entries follow this one. An implementation MUST use this value to determine the location of the next entry (if multiple entries are present in a buffer).

FileIndex (4 bytes): A 32-bit unsigned integer that contains the byte offset of the file within the parent directory. For file systems in which the position of a file within the parent directory is not fixed and can be changed at any time to maintain sort order, this field SHOULD be set to 0, and MUST be ignored.<86>

CreationTime (8 bytes): The time when the file was created; see section 2.1.1. The value of this field MUST be greater than or equal to 0.

LastAccessTime (8 bytes): The last time the file was accessed; see section 2.1.1. The value of this field MUST be greater than or equal to 0.

LastWriteTime (8 bytes): The last time information was written; see section 2.1.1. The value of this field MUST be greater than or equal to 0.

ChangeTime (8 bytes): The last time the file was changed; see section 2.1.1. The value of this field MUST be greater than or equal to 0.

EndOfFile (8 bytes): A 64-bit signed integer that contains the absolute new end-of-file position as a byte offset from the start of the file. EndOfFile specifies the offset to the byte immediately following the last valid byte in the file. Because this value is zero-based, it actually refers to the first free byte in the file. That is, it is the offset from the beginning of the file at which new bytes appended to the file will be written. The value of this field MUST be greater than or equal to 0.

AllocationSize (8 bytes): A 64-bit signed integer that contains the file allocation size, in bytes. The value of this field MUST be an integer multiple of the cluster size.

FileAttributes (4 bytes): A 32-bit unsigned integer that contains the file attributes. Valid attributes are as specified in section 2.6.

FileNameLength (4 bytes): A 32-bit unsigned integer that specifies the length, in bytes, of the file name contained within the **FileName** member.

EaSize (4 bytes): A 32-bit unsigned integer that contains the combined length, in bytes, of the extended attributes (EA) for the file.

ShortNameLength (1 byte): A 8-bit signed integer that specifies the length, in bytes, of the file name contained within the **ShortName** member.

Reserved1 (1 byte): An 8-bit field. This field is reserved. This field MUST be set to zero, and MUST be ignored.

ShortName (24 bytes): A sequence of Unicode characters containing the short (8.3) file name. When working with this field, use **ShortNameLength** to determine the length of the file name rather than assuming the presence of a trailing null delimiter.

Reserved2 (2 bytes): A 16-bit field. This field is reserved. This field MUST be set to zero, and MUST be ignored.

FileId (8 bytes): An 8-byte file reference number for the file. This number SHOULD be generated and assigned to the file by the file system. For file systems which do not support FileId, this field MUST be set to 0, and MUST be ignored.

FileName (variable): A sequence of Unicode characters containing the file name. When working with this field, use **FileNameLength** to determine the length of the file name rather than assuming the presence of a trailing null delimiter. Dot directory names are valid for this field. For more details, see section 2.1.5.1.

This operation returns a status code, as specified in [MS-ERREF] section 2.3. The status code returned directly by the function that processes this file information class MUST be STATUS_SUCCESS or one of the following.

Error code	Meaning
STATUS_INFO_LENGTH_MISMATCH 0xC0000004	The specified information record length does not match the length that is required for the specified information class.

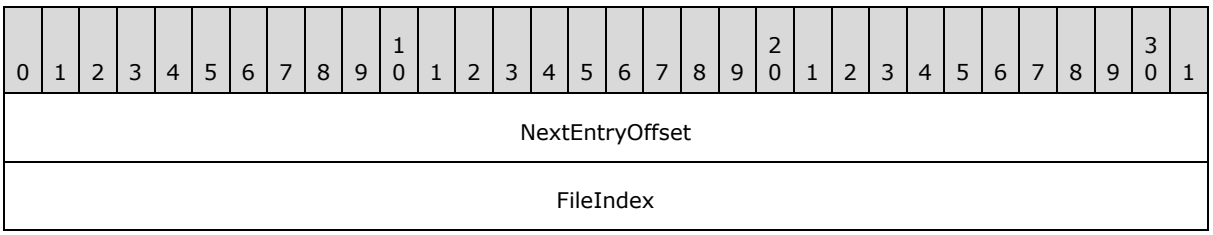
2.4.18 FileIdFullDirectoryInformation

This information class is used in directory enumeration to return detailed information about the contents of a directory.

This information class returns a list that contains a **FILE_ID_FULL_DIR_INFORMATION** data element for each file or directory within the target directory. This list MUST reflect the presence of a subdirectory named "." (synonymous with the target directory itself) within the target directory and one named ".." (synonymous with the parent directory of the target directory). For more details, see section 2.1.5.1.

When multiple **FILE_ID_FULL_DIR_INFORMATION** data elements are present in the buffer, each MUST be aligned on an 8-byte boundary. Any bytes inserted for alignment SHOULD be set to zero, and the receiver MUST ignore them. No padding is required following the last data element.

A **FILE_ID_FULL_DIR_INFORMATION** data element is as follows.



CreationTime
...
LastAccessTime
...
LastWriteTime
...
ChangeTime
...
EndOfFile
...
AllocationSize
...
FileAttributes
FileNameLength
EaSize
Reserved
FileId
...
FileName (variable)
...

NextEntryOffset (4 bytes): A 32-bit unsigned integer that contains the byte offset from the beginning of this entry, at which the next FILE_ID_FULL_DIR_INFORMATION entry is located, if multiple entries are present in a buffer. This field SHOULD<87> be zero if no other entries follow this one. An implementation MUST use this value to determine the location of the next entry (if multiple entries are present in a buffer).

FileIndex (4 bytes): A 32-bit unsigned integer that contains the byte offset of the file within the parent directory. For file systems in which the position of a file within the parent directory is not fixed and can be changed at any time to maintain sort order, this field SHOULD be set to 0 and MUST be ignored.<88>

CreationTime (8 bytes): The time when the file was created; see section 2.1.1. The value of this field MUST be greater than or equal to 0.

LastAccessTime (8 bytes): The last time the file was accessed; see section 2.1.1. The value of this field MUST be greater than or equal to 0.

LastWriteTime (8 bytes): The last time information was written; see section 2.1.1. The value of this field MUST be greater than or equal to 0.

ChangeTime (8 bytes): The last time the file was changed; see section 2.1.1. The value of this field MUST be greater than or equal to 0.

EndOfFile (8 bytes): A 64-bit signed integer that contains the absolute new end-of-file position as a byte offset from the start of the file. EndOfFile specifies the offset to the byte immediately following the last valid byte in the file. Because this value is zero-based, it actually refers to the first free byte in the file. That is, it is the offset from the beginning of the file at which new bytes appended to the file will be written. The value of this field MUST be greater than or equal to 0.

AllocationSize (8 bytes): A 64-bit signed integer that contains the file allocation size, in bytes. The value of this field MUST be an integer multiple of the cluster size.

FileAttributes (4 bytes): A 32-bit unsigned integer that contains the file attributes. Valid attributes are as specified in section 2.6.

FileNameLength (4 bytes): A 32-bit unsigned integer that specifies the length, in bytes, of the file name contained within the **FileName** member.

EaSize (4 bytes): A 32-bit unsigned integer that contains the combined length, in bytes, of the extended attributes (EA) for the file.

Reserved (4 bytes): Reserved for alignment. This field can contain any value and MUST be ignored.

FileId (8 bytes): An 8-byte file reference number for the file. This number is generated and assigned to the file by the file system. For file systems which do not support FileId, this field MUST be set to 0, and MUST be ignored.

FileName (variable): A sequence of Unicode characters containing the file name. When working with this field, use **FileNameLength** to determine the length of the file name rather than assuming the presence of a trailing null delimiter. Dot directory names are valid for this field. For more details, see section 2.1.5.1.

This operation returns a status code, as specified in [MS-ERREF] section 2.3. The status code returned directly by the function that processes this file information class MUST be STATUS_SUCCESS or one of the following.

Error code	Meaning
STATUS_INFO_LENGTH_MISMATCH 0xC0000004	The specified information record length does not match the length that is required for the specified information class.

2.4.19 FileIdGlobalTxDirectoryInformation

This information class is used locally to query transactional visibility information for the files in a directory. This information class MAY be implemented for file systems that return the FILE_SUPPORTS_TRANSACTIONS flag in response to **FileFsAttributeInformation** specified in section 2.5.1. This information class MUST NOT be implemented for file systems that do not return that flag.

When multiple **FILE_ID_GLOBAL_TX_DIR_INFORMATION** data elements are present in the buffer, each MUST be aligned on an 8-byte boundary. Any bytes inserted for alignment SHOULD be set to zero, and the receiver MUST ignore them. No padding is required following the last data element.

A **FILE_ID_GLOBAL_TX_DIR_INFORMATION** data element is as follows.

0	1	2	3	4	5	6	7	8	9	10	1	2	3	4	5	6	7	8	9	20	1	2	3	4	5	6	7	8	9	30	1
NextEntryOffset																															
FileIndex																															
CreationTime																															
...																															
LastAccessTime																															
...																															
LastWriteTime																															
...																															
ChangeTime																															
...																															
EndOfFile																															
...																															
AllocationSize																															
...																															
FileAttributes																															
FileNameLength																															

FileId
...
LockingTransactionId
...
...
...
TxInfoFlags
FileName (variable)
...

NextEntryOffset (4 bytes): A 32-bit unsigned integer that contains the byte offset from the beginning of this entry, at which the next FILE_ID_GLOBAL_TX_DIR_INFORMATION entry is located, if multiple entries are present in a buffer. This member MUST be zero if no other entries follow this one. An implementation MUST use this value to determine the location of the next entry (if multiple entries are present in a buffer).

FileIndex (4 bytes): A 32-bit unsigned integer that contains the byte offset of the file within the parent directory. For file systems in which the position of a file within the parent directory is not fixed and can be changed at any time to maintain sort order, this field SHOULD be set to 0 and MUST be ignored.

CreationTime (8 bytes): The time when the file was created; see section 2.1.1. The value of this field MUST be greater than or equal to 0.

LastAccessTime (8 bytes): The last time the file was accessed; see section 2.1.1. The value of this field MUST be greater than or equal to 0.

LastWriteTime (8 bytes): The last time information was written to the file; see section 2.1.1. The value of this field MUST be greater than or equal to 0.

ChangeTime (8 bytes): The last time the file was changed; see section 2.1.1. The value of this field MUST be greater than or equal to 0.

EndOfFile (8 bytes): A 64-bit signed integer that contains the absolute new end-of-file position as a byte offset from the start of the file. **EndOfFile** specifies the offset to the byte immediately following the last valid byte in the file. Because this value is zero-based, it actually refers to the first free byte in the file. That is, it is the offset from the beginning of the file at which new bytes appended to the file will be written. The value of this field MUST be greater than or equal to 0.

AllocationSize (8 bytes): A 64-bit signed integer that contains the file allocation size, in bytes. The value of this field MUST be an integer multiple of the cluster size.

FileAttributes (4 bytes): A 32-bit unsigned integer that contains the file attributes. Valid attributes are as specified in section 2.6.

FileNameLength (4 bytes): A 32-bit unsigned integer that specifies the length, in bytes, of the file name contained within the **FileName** member.

FileId (8 bytes): An 8-byte file reference number for the file. This number is generated and assigned to the file by the file system. For file systems that do not support **FileId**, this field MUST be set to 0, and MUST be ignored.

LockingTransactionId (16 bytes): A **GUID** value that is the ID of the transaction that has this file locked for modification. This number is generated and assigned by the file system. If the FILE_ID_GLOBAL_TX_DIR_INFO_FLAG_WRITELOCKED flag is not set in the **TxInfoFlags** field, this field MUST be ignored.

TxInfoFlags (4 bytes): A 32-bit unsigned integer that contains a bitmask of flags that indicate the transactional visibility of the file. The value of this field MUST be a bitwise OR of zero or more of the following values. Any flag values not explicitly mentioned here can be set to any value and MUST be ignored. If the FILE_ID_GLOBAL_TX_DIR_INFO_FLAG_WRITELOCKED flag is not set, the other flags MUST NOT be set. If flags other than FILE_ID_GLOBAL_TX_DIR_INFO_FLAG_WRITELOCKED are set, FILE_ID_GLOBAL_TX_DIR_INFO_FLAG_WRITELOCKED MUST be set.

Value	Meaning
FILE_ID_GLOBAL_TX_DIR_INFO_FLAG_WRITELOCKED 0x00000001	The file is locked for modification by a transaction. The transaction's ID MUST be contained in the LockingTransactionId field if this flag is set.
FILE_ID_GLOBAL_TX_DIR_INFO_FLAG_VISIBLE_TO_TX 0x00000002	The file is visible to transacted enumerators of the directory whose transaction ID is in the LockingTransactionId field.
FILE_ID_GLOBAL_TX_DIR_INFO_FLAG_VISIBLE_OUTSIDE_TX 0x00000004	The file is visible to transacted enumerators of the directory other than the one whose transaction ID is in the LockingTransactionId field, and it is visible to non-transacted enumerators of the directory.

FileName (variable): A sequence of Unicode characters containing the file name. When working with this field, use **FileNameLength** to determine the length of the file name rather than assuming the presence of a trailing null delimiter.

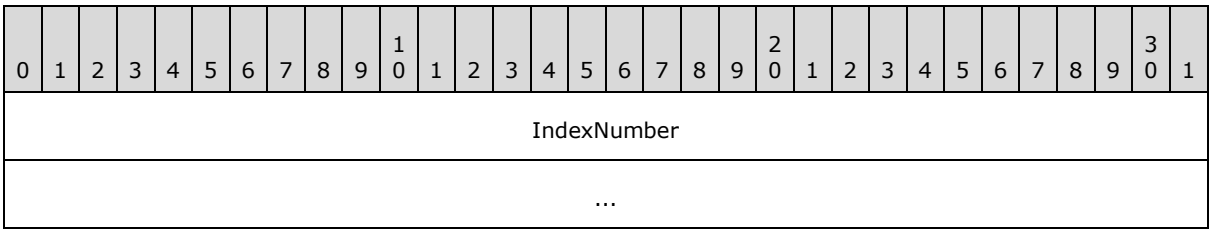
This operation returns a status code, as specified in [MS-ERREF] section 2.3. The status code returned directly by the function that processes this file information class MUST be STATUS_SUCCESS or one of the following.

Error code	Meaning
STATUS_NOT_SUPPORTED 0xC00000BB	The request is not supported.

2.4.20 FileInternalInformation

This information class is used to query for the file system's 8-byte file reference number for a file.

A **FILE_INTERNAL_INFORMATION** data element, defined as follows, is returned by the server.



IndexNumber (8 bytes): A 64-bit signed integer that contains the 8-byte file reference number for the file. This number MUST be assigned by the file system and is unique to the volume on which the file or directory is located. This file reference number is the same as the file reference number that is stored in the **FileId** field of the **FILE_ID_BOTH_DIR_INFORMATION** and **FILE_ID_FULL_DIR_INFORMATION** data elements. The value of this field MUST be greater than or equal to 0. For file systems which do not support a file reference number, this field MUST be set to 0, and MUST be ignored.

This operation returns a status code, as specified in [MS-ERREF] section 2.3. The status code returned directly by the function that processes this file information class MUST be STATUS_SUCCESS or one of the following.

Error code	Meaning
STATUS_INFO_LENGTH_MISMATCH 0xC0000004	The specified information record length does not match the length that is required for the specified information class.

2.4.21 FileLinkInformation

This information class is used to create a hard link to an existing file.<89> The Server Message Block (SMB) Protocol [MS-SMB] and the Server Message Block (SMB) Version 2 Protocol [MS-SMB2] implement unique structure variants:

- **FILE_LINK_INFORMATION_TYPE_1**, as specified in section 2.4.21.1.
- **FILE_LINK_INFORMATION_TYPE_2**, as specified in section 2.4.21.2.

This operation returns a status code, as specified in [MS-ERREF] section 2.3. The status code returned directly by the function that processes this file information class MUST be STATUS_SUCCESS or one of the following.

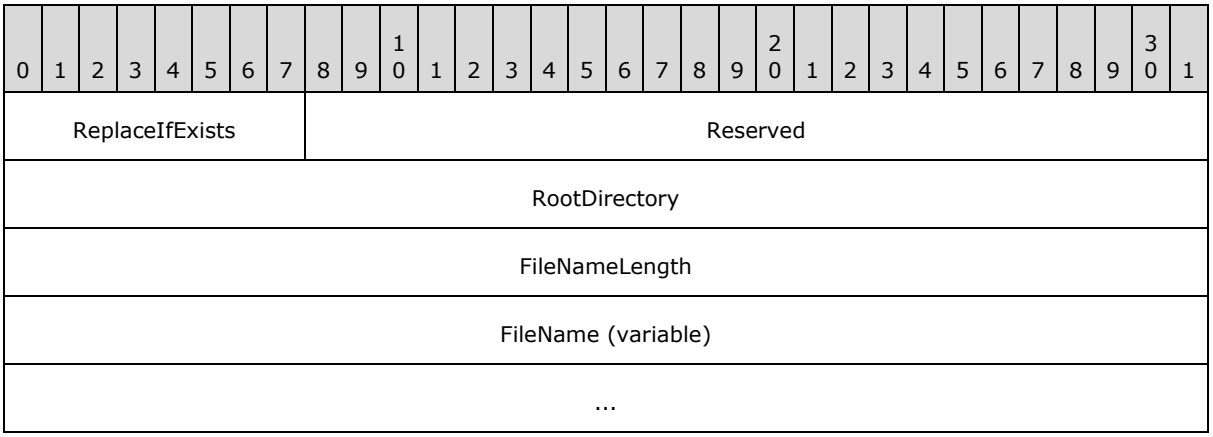
Error code	Meaning
STATUS_INVALID_PARAMETER 0xC000000D	An invalid parameter was specified for the **RootDirectory** field.
STATUS_FILE_IS_A_DIRECTORY 0xC00000BA	The file that was specified is a directory.
STATUS_ACCESS_DENIED	The object has been deleted.

Error code	Meaning
0xC0000022	
STATUS_OBJECT_NAME_INVALID 0xC0000033	The object name is invalid for the target file system.
STATUS_TOO_MANY_LINKS 0xC0000265	An attempt was made to create more links on a file than the file system supports.
STATUS_OBJECT_NAME_COLLISION 0xC0000035	The specified name already exists and **ReplaceIfExists** is zero.
STATUS_INFO_LENGTH_MISMATCH 0xC0000004	The specified information record length does not match the length that is required for the specified information class.

2.4.21.1 FileLinkInformation for the SMB Protocol

This information class is used to create a hard link to an existing file via the SMB Protocol as specified in [MS-SMB].

A **FILE_LINK_INFORMATION_TYPE_1** data element, defined as follows, is provided by the client.



ReplaceIfExists (1 byte): A Boolean (section 2.1.8) value. Set to TRUE to indicate that if the link already exists, it SHOULD be replaced with the new link. Set to FALSE to indicate that the link creation operation MUST fail if the link already exists.

Reserved (3 bytes): This field SHOULD be set to zero by the client and MUST be ignored by the server.

RootDirectory (4 bytes): A 32-bit unsigned integer that contains the file handle for the directory where the link is to be created. For network operations, this value MUST always be zero.

FileNameLength (4 bytes): A 32-bit unsigned integer that contains the length in bytes of the **FileName** field.

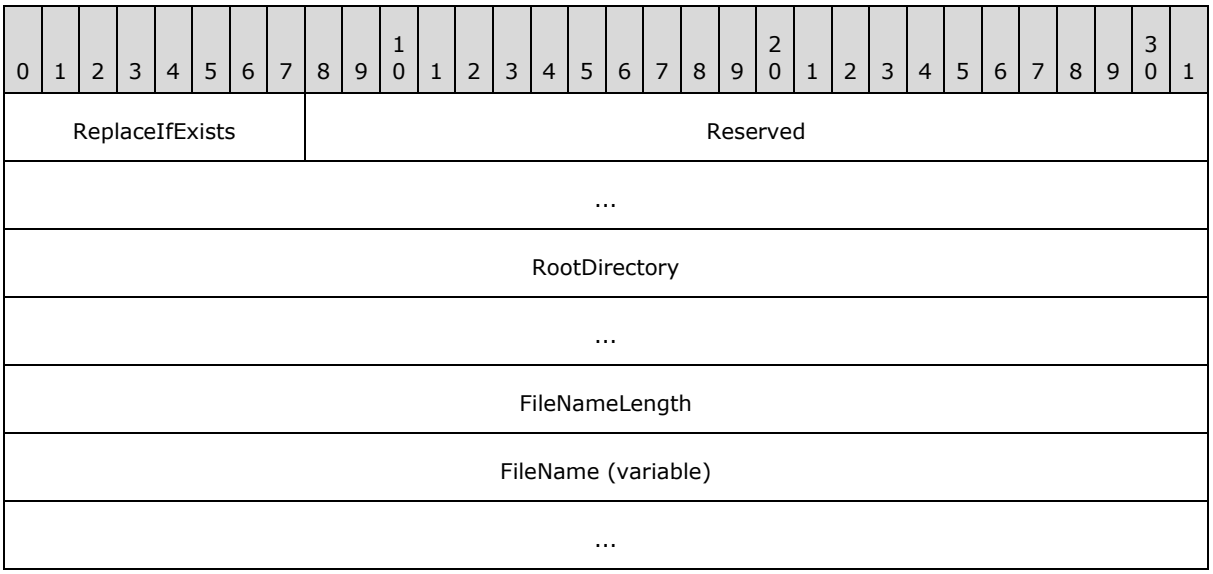
FileName (variable): A sequence of Unicode characters that contains the name to be assigned to the newly created link. When working with the **FileName** field, the **FileNameLength** field is used to determine the length of the file name rather than assuming the presence of a

trailing null delimiter. If the **RootDirectory** field is zero, this field MUST specify a full pathname to the link to be created. For network operations, this pathname is relative to the root of the share. If the **RootDirectory** field is not zero, this field MUST specify a pathname, relative to **RootDirectory**, for the link name.

2.4.21.2 FileLinkInformation for the SMB2 Protocol

This information class is used to create a hard link to an existing file via the SMB Version 2 Protocol, as specified in [MS-SMB2].

A **FILE_LINK_INFORMATION_TYPE_2** data element, defined as follows, is provided by the client.



ReplaceIfExists (1 byte): A Boolean (section 2.1.8) value. Set to TRUE to indicate that if the link already exists, it SHOULD be replaced with the new link. Set to FALSE to indicate that the link creation operation MUST fail if the link already exists.

Reserved (7 bytes): Reserved for alignment. This field can contain any value and MUST be ignored.

RootDirectory (8 bytes): A 64-bit unsigned integer that contains the file handle for the directory where the link is to be created. For network operations, this value MUST be zero.

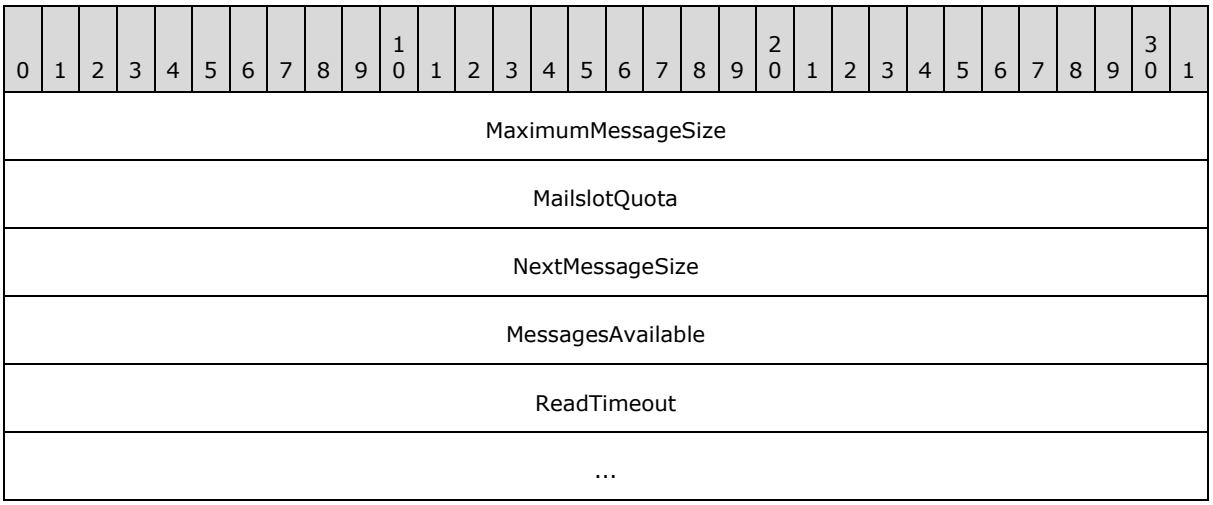
FileNameLength (4 bytes): A 32-bit unsigned integer that specifies the length in bytes of the file name contained within the **FileName** field.

FileName (variable): A sequence of Unicode characters containing the name to be assigned to the newly created link. When working with this field, the **FileNameLength** field is used to determine the length of the file name rather than assuming the presence of a trailing null delimiter. If the **RootDirectory** field is zero, this field MUST specify a full pathname to the link to be created. For network operations, this pathname is relative to the root of the share. If the **RootDirectory** field is not zero, this field MUST specify a pathname, relative to **RootDirectory**, for the link name.

2.4.22 FileMailslotQueryInformation

This information class is used locally to query information on a mailslot.

A **FILE_MAILSLOT_QUERY_INFORMATION** data element, defined as follows, is returned to the caller.



MaximumMessageSize (4 bytes): A 32-bit unsigned integer that contains the maximum size of a single message that can be written to the mailslot, in bytes. To specify that the message can be of any size, set this value to zero.

MailslotQuota (4 bytes): A 32-bit unsigned integer that contains the quota, in bytes, for the mailslot. The mailslot quota specifies the in-memory pool quota that is reserved for writes to this mailslot.

NextMessageSize (4 bytes): A 32-bit unsigned integer that contains the next message size, in bytes.

MessagesAvailable (4 bytes): A 32-bit unsigned integer that contains the total number of messages waiting to be read from the mailslot.

ReadTimeout (8 bytes): A 64-bit signed integer that contains the time a read operation can wait for a message to be written to the mailslot before a time-out occurs in milliseconds. The value of this field MUST be (-1) or greater than or equal to 0. A value of (-1) requests that the read wait forever for a message, without timing out. A value of 0 requests that the read not wait and return immediately whether a pending message is available to be read or not.

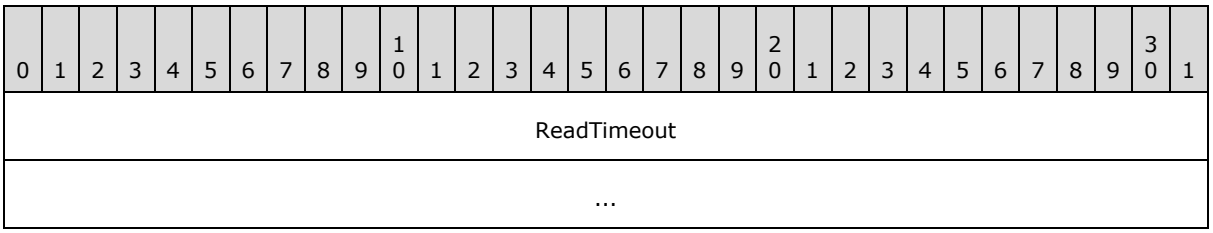
This operation returns a status code, as specified in [MS-ERREF] section 2.3. The status code returned directly by the function that processes this file information class MUST be STATUS_SUCCESS or one of the following.

Error code	Meaning
STATUS_INFO_LENGTH_MISMATCH 0xC0000004	The specified information record length does not match the length that is required for the specified information class.

2.4.23 FileMailslotSetInformation

This information class is used locally to set information on a mailslot.

A **FILE_MAILSLOT_SET_INFORMATION** data element, defined as follows, is provided by the caller.



ReadTimeout (8 bytes): A 64-bit signed integer that contains the time that a read operation can wait for a message to be written to the mailslot before a time-out occurs as follows:

- A positive value specifies the operation time-out as an absolute system time on the server, represented as a count of 100-nanosecond intervals since January 1, 1601.
- A negative value specifies the number of 100-nanosecond intervals for the operation to time out relative to the current server time.
- A value of -1 (0xFFFFFFFFFFFFFFFF) requests that the read wait forever for a message without timing out.
- A value of zero sends a request that the read not wait and return immediately, whether a pending message is available to be read or not.

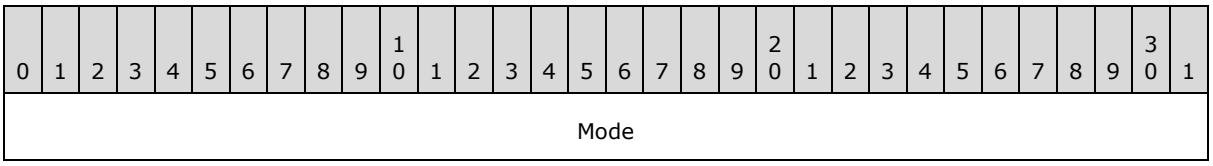
This operation returns a status code, as specified in [MS-ERREF] section 2.3. The status code returned directly by the function that processes this file information class MUST be STATUS_SUCCESS or one of the following.

Error code	Meaning
STATUS_INFO_LENGTH_MISMATCH 0xC0000004	The specified information record length does not match the length that is required for the specified information class.

2.4.24 FileModeInformation

This information class is used to query or set the mode of the file. The mode returned by a query corresponds to the **CreateOptions** used in the initial create operation, modified by any set FileModeInformation operations performed since the create operation.<90>

A **FILE_MODE_INFORMATION** data element, defined as follows, is returned by the server or provided by the client.



Mode (4 bytes): A 32-bit unsigned integer that specifies how the file will subsequently be accessed.

Value	Meaning
FILE_WRITE_THROUGH 0x00000002	When set, any system services, file system drivers (FSDs), and drivers that write data to the file must actually transfer the data into the file before any requested write operation is considered complete.
FILE_SEQUENTIAL_ONLY 0x00000004	This is a hint that informs the cache that it SHOULD<91> optimize for sequential access. Non-sequential access of the file may result in performance degradation.
FILE_NO_INTERMEDIATE_BUFFERING 0x00000008	When set, the file cannot be cached or buffered in a driver's internal buffers.
FILE_SYNCHRONOUS_IO_ALERT 0x00000010	When set, all operations on the file are performed synchronously. Any wait on behalf of the caller is subject to premature termination from alerts. This flag also causes the I/O system to maintain the file position context.
FILE_SYNCHRONOUS_IO_NONALERT 0x00000020	When set, all operations on the file are performed synchronously. Wait requests in the system to synchronize I/O queuing and completion are not subject to alerts. This flag also causes the I/O system to maintain the file position context.
FILE_DELETE_ON_CLOSE 0x00001000	This flag is not implemented and is always returned as not set.

This operation returns a status code, as specified in [MS-ERREF] section 2.3. The status code returned directly by the function that processes this file information class MUST be STATUS_SUCCESS or one of the following.

Error code	Meaning
STATUS_INFO_LENGTH_MISMATCH 0xC0000004	The specified information record length does not match the length that is required for the specified information class.
STATUS_INVALID_PARAMETER	An attempt to set the file mode returns STATUS_INVALID_PARAMETER in any of the following cases: ▪ The mode field contains any flag other than FILE_WRITE_THROUGH, FILE_SEQUENTIAL_ONLY, FILE_SYNCHRONOUS_IO_ALERT, or FILE_SYNCHRONOUS_IO_NONALERT. ▪ FILE_SYNCHRONOUS_IO_ALERT or FILE_SYNCHRONOUS_IO_NONALERT is set and the file was not opened for synchronous I/O. ▪ Neither FILE_SYNCHRONOUS_IO_ALERT nor FILE_SYNCHRONOUS_IO_NONALERT are set and the file was opened for synchronous I/O. ▪ FILE_SYNCHRONOUS_IO_ALERT and

Error code	Meaning
	FILE_SYNCHRONOUS_IO_NONALERT are both set.

2.4.25 FileNameInformation

This information class is used locally to query the name of a file. This information class returns a **FILE_NAME_INFORMATION** data element as specified in section 2.1.7.

This operation returns a status code, as specified in [MS-ERREF] section 2.3. The status code returned directly by the function that processes this file information class MUST be STATUS_SUCCESS or one of the following.

Error code	Meaning
STATUS_NOT_SUPPORTED 0xC00000BB	The resource is not supported.
STATUS_BUFFER_OVERFLOW 0x80000005	The output buffer was filled before the complete name could be returned.

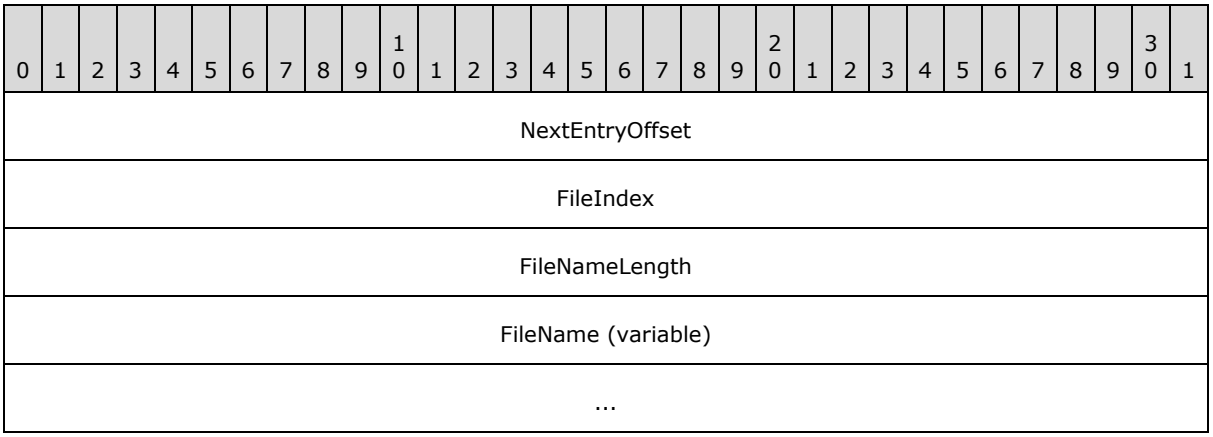
2.4.26 FileNamesInformation

This information class is used in directory enumeration to return detailed information about the contents of a directory.

This information class returns a list that contains a **FILE_NAMES_INFORMATION** data element for each file or directory within the target directory. This list MUST reflect the presence of a subdirectory named "." (synonymous with the target directory itself) within the target directory and one named ".." (synonymous with the parent directory of the target directory). For more details, see section 2.1.5.1.

When multiple **FILE_NAMES_INFORMATION** data elements are present in the buffer, each MUST be aligned on an 8-byte boundary. Any bytes inserted for alignment SHOULD be set to zero, and the receiver MUST ignore them. No padding is required following the last data element.

A **FILE_NAMES_INFORMATION** data element is as follows.



NextEntryOffset (4 bytes): A 32-bit unsigned integer that contains the byte offset from the beginning of this entry, at which the next FILE_NAMES_INFORMATION entry is located, if multiple entries are present in a buffer. This member MUST be zero if no other entries follow this one. An implementation MUST use this value to determine the location of the next entry (if multiple entries are present in a buffer).

FileIndex (4 bytes): A 32-bit unsigned integer that contains the byte offset of the file within the parent directory. For file systems in which the position of a file within the parent directory is not fixed and can be changed at any time to maintain sort order, this field SHOULD be set to 0, and MUST be ignored.<92>

FileNameLength (4 bytes): A 32-bit unsigned integer that specifies the length, in bytes, of the file name contained within the **FileName** member.

FileName (variable): A sequence of Unicode characters containing the file name. When working with this field, use **FileNameLength** to determine the length of the file name rather than assuming the presence of a trailing null delimiter.

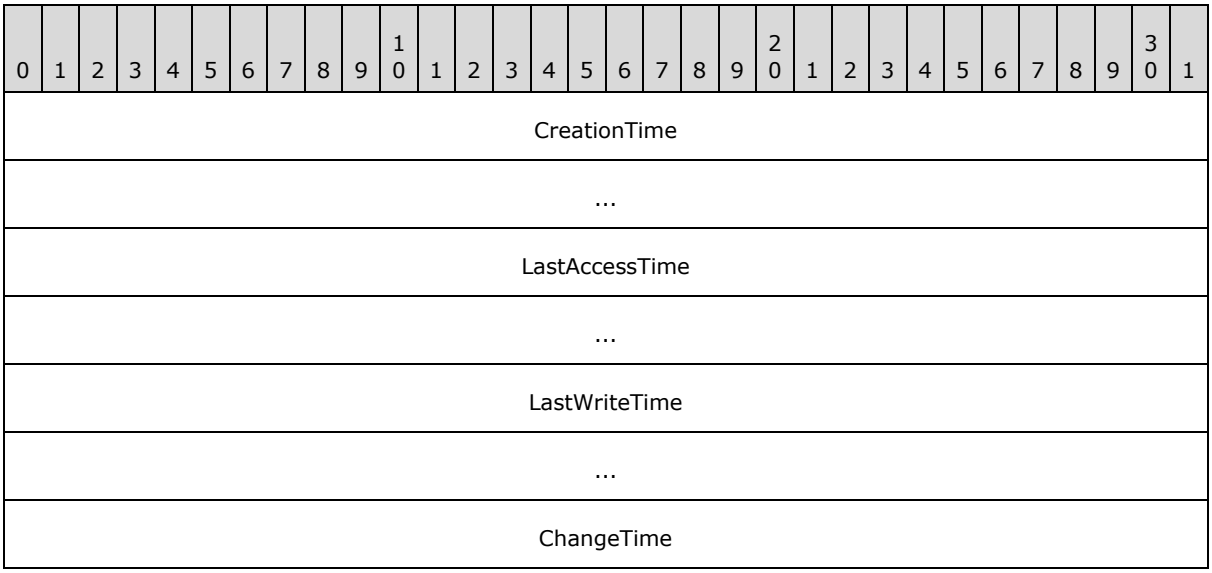
This operation returns a status code, as specified in [MS-ERREF] section 2.3. The status code returned directly by the function that processes this file information class MUST be STATUS_SUCCESS or one of the following.

Error code	Meaning
STATUS_INFO_LENGTH_MISMATCH 0xC0000004	The specified information record length does not match the length that is required for the specified information class.

2.4.27 FileNetworkOpenInformation

This information class is used to query for information that is commonly needed when a file is opened across a network.<93>

A **FILE_NETWORK_OPEN_INFORMATION** data element, defined as follows, is returned by the server.



...
AllocationSize
...
EndOfFile
...
FileAttributes
Reserved

CreationTime (8 bytes): The time when the file was created; see section 2.1.1. The value of this field MUST be greater than or equal to 0.

LastAccessTime (8 bytes): The last time the file was accessed; see section 2.1.1. The value of this field MUST be greater than or equal to 0.

LastWriteTime (8 bytes): The last time information was written to the file; see section 2.1.1. The value of this field MUST be greater than or equal to 0.

ChangeTime (8 bytes): The last time the file was changed; see section 2.1.1. The value of this field MUST be greater than or equal to 0.

AllocationSize (8 bytes): A 64-bit signed integer that contains the file allocation size, in bytes. The value of this field MUST be an integer multiple of the cluster size.

EndOfFile (8 bytes): A 64-bit signed integer that contains the absolute new end-of-file position as a byte offset from the start of the file. EndOfFile specifies the offset to the byte immediately following the last valid byte in the file. Because this value is zero-based, it actually refers to the first free byte in the file. That is, it is the offset from the beginning of the file at which new bytes appended to the file will be written. The value of this field MUST be greater than or equal to 0.

FileAttributes (4 bytes): A 32-bit unsigned integer that contains the file attributes. Valid attributes are as specified in section 2.6.

Reserved (4 bytes): A 32-bit field. This field is reserved. This field can be set to any value, and MUST be ignored.

This operation returns a status code, as specified in [MS-ERREF] section 2.3. The status code returned directly by the function that processes this file information class MUST be STATUS_SUCCESS or one of the following.

Error code	Meaning
STATUS_INFO_LENGTH_MISMATCH 0xC0000004	The specified information record length does not match the length that is required for the specified information class.
STATUS_ACCESS_DENIED	The handle was not opened to read file data or file attributes.

Error code	Meaning
0xC0000022	

2.4.28 FileObjectIdInformation

This information class is used locally to query object ID information for the files in a directory on a volume. The query MUST fail if the file system does not support object IDs.<94>

The data returned to the caller will take one of two forms. The choice of which data structure to use, and the interpretation of the data within it, is application-specific. An application implementer chooses one of the following two data elements as the structure for its object ID information data.<95>

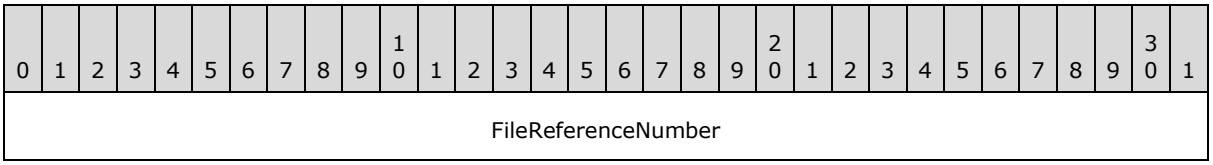
- FILE_OBJECTID_INFORMATION_TYPE_1 (section 2.4.28.1).
- FILE_OBJECTID_INFORMATION_TYPE_2 (section 2.4.28.2).

This operation returns a status code, as specified in [MS-ERREF] section 2.3. The status code returned directly by the function that processes this file information class MUST be STATUS_SUCCESS or one of the following.

Error code	Meaning
STATUS_INVALID_DEVICE_REQUEST 0xC0000010	The target file system does not implement this functionality.
STATUS_INVALID_INFO_CLASS 0xC0000003	The specified information class is not a valid information class for the specified object.
STATUS_INFO_LENGTH_MISMATCH 0xC0000004	The specified information record length does not match the length that is required for the specified information class.
STATUS_INVALID_PARAMETER 0xC000000D	The file specified is not a valid parameter.
STATUS_NO_SUCH_FILE 0xC000000F	The file does not exist.
STATUS_NO_MORE_FILES 0x80000006	No more files were found which match the file specification.
STATUS_BUFFER_OVERFLOW 0x80000005	The output buffer was filled before all of the **ObjectID** information could be returned. Only complete **FILE_OBJECTID_INFORMATION** structures are returned.

2.4.28.1 FILE_OBJECTID_INFORMATION_TYPE_1

A **FILE_OBJECTID_INFORMATION_TYPE_1** data element is as follows.



...
ObjectId
...
...
...
BirthVolumeId
...
...
...
BirthObjectId
...
...
...
DomainId
...
...
...

FileReferenceNumber (8 bytes): A 64-bit unsigned integer that contains the file reference number for the file. NTFS generates this number and assigns it to the file automatically when the file is created. The file reference number is unique within the volume on which the file exists.

ObjectId (16 bytes): A 16-byte GUID that uniquely identifies the file or directory within the volume on which it resides. Specifically, the same object ID can be assigned to another file or directory on a different volume, but it MUST NOT be assigned to another file or directory on the same volume.

BirthVolumeId (16 bytes): A 16-byte 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

may not be the same as the object identifier of the volume on which the object presently resides.

BirthObjectId (16 bytes): A 16-byte 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 may not be the same as the **ObjectId** member at present.<96>

DomainId (16 bytes): A 16-byte GUID value containing the domain identifier. This value is unused; it SHOULD be zero, and MUST be ignored.

2.4.28.2 FILE_OBJECTID_INFORMATION_TYPE_2

A **FILE_OBJECTID_INFORMATION_TYPE_2** data element is as follows.

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
FileReferenceNumber																															
...																															
ObjectId																															
...																															
...																															
...																															
ExtendedInfo																															
...																															
...																															
...																															
...																															
...																															
...																															
...																															
(ExtendedInfo cont'd for 4 rows)																															

FileReferenceNumber (8 bytes): A 64-bit unsigned integer that contains the file reference number for the file. NTFS generates this number and assigns it to the file automatically when

the file is created. The file reference number is unique within the volume on which the file exists.

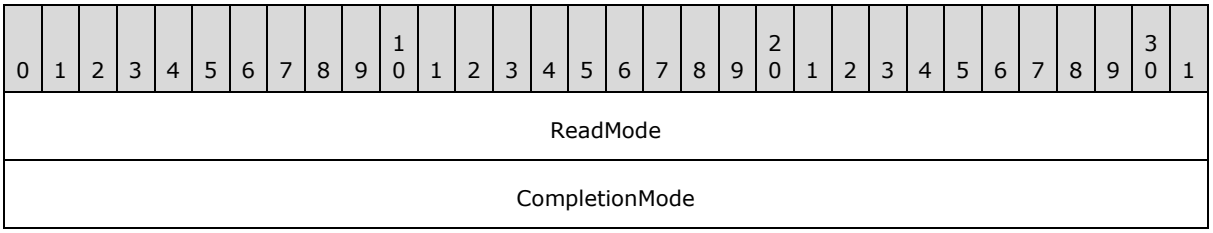
ObjectId (16 bytes): A 16-byte GUID that uniquely identifies the file or directory within the volume on which it resides. Specifically, the same object ID can be assigned to another file or directory on a different volume, but it MUST NOT be assigned to another file or directory on the same volume.

ExtendedInfo (48 bytes): A 48-byte BLOB that contains application-specific extended information on the file object. If no extended information has been written for this file, the server MUST return 48 bytes of 0x00 in this field.

2.4.29 FilePipeInformation

This information class is used to query or set information on a named pipe that is not specific to one end of the pipe or another.

A **FILE_PIPE_INFORMATION** data element, defined as follows, is returned by the server or provided by the client.



ReadMode (4 bytes): A 32-bit unsigned integer that MUST contain one of the following values.

Value	Meaning
FILE_PIPE_BYTE_STREAM_MODE 0x00000000	If this value is specified, data MUST be read from the pipe as a stream of bytes.
FILE_PIPE_MESSAGE_MODE 0x00000001	If this value is specified, data MUST be read from the pipe as a stream of messages.

If this field is set to FILE_PIPE_BYTE_STREAM_MODE, any attempt to subsequently change it MUST fail with a STATUS_INVALID_PARAMETER error code.

CompletionMode (4 bytes): A 32-bit unsigned integer that MUST contain one of the following values.

Value	Meaning
FILE_PIPE_QUEUE_OPERATION 0x00000000	If this value is specified, blocking mode MUST be enabled. When the pipe is being connected, read to, or written from, the operation is not completed until there is data to read, all data is written, or a client is connected. Use of this mode can result in the server waiting indefinitely for a client process to perform an action.
FILE_PIPE_COMPLETE_OPERATION 0x00000001	If this value is specified, non-blocking mode MUST be enabled. When the pipe is being connected, read to, or

Value	Meaning
	written from, the operation completes immediately.

This operation returns a status code, as specified in [MS-ERREF] section 2.3. The status code returned directly by the function that processes this file information class MUST be STATUS_SUCCESS or one of the following.

Error code	Meaning
STATUS_INFO_LENGTH_MISMATCH 0xC0000004	The specified information record length does not match the length that is required for the specified information class.
STATUS_INVALID_PARAMETER 0xC000000D	An invalid parameter was passed to a service or function. When setting the FilePipeInformation information level, STATUS_INVALID_PARAMETER will be returned: ▪ If the **ReadMode** field is set to FILE_PIPE_BYTE_STREAM_MODE and a subsequent set operation attempts to set the **ReadMode** field to any value other than FILE_PIPE_BYTE_STREAM_MODE. ▪ If the value of the **ReadMode** field is not equal to FILE_PIPE_MESSAGE_MODE or FILE_PIPE_BYTE_STREAM_MODE. ▪ If the value of the **CompletionMode** field is not equal to FILE_PIPE_QUEUE_OPERATION or FILE_PIPE_COMPLETE_OPERATION.

For more information on named pipes, please see [PIPE].

2.4.30 FilePipeLocalInformation

This information class is used to query information on a named pipe that is associated with the end of the pipe that is being queried.

A **FILE_PIPE_LOCAL_INFORMATION** data element, defined as follows, is returned by the server.

0	1	2	3	4	5	6	7	8	9	10	1	2	3	4	5	6	7	8	9	20	1	2	3	4	5	6	7	8	9	30	1
NamedPipeType																															
NamedPipeConfiguration																															
MaximumInstances																															
CurrentInstances																															
InboundQuota																															
ReadDataAvailable																															

OutboundQuota
WriteQuotaAvailable
NamedPipeState
NamedPipeEnd

NamedPipeType (4 bytes): A 32-bit unsigned integer that contains the named pipe type. MUST be one of the following.

Value	Meaning
FILE_PIPE_BYTE_STREAM_TYPE 0x00000000	If this value is specified, data MUST be read from the pipe as a stream of bytes.
FILE_PIPE_MESSAGE_TYPE 0x00000001	If this flag is specified, data MUST be read from the pipe as a stream of messages.

NamedPipeConfiguration (4 bytes): A 32-bit unsigned integer that contains the named pipe configuration. MUST be one of the following.

Value	Meaning
FILE_PIPE_INBOUND 0x00000000	If this value is specified, the flow of data in the pipe goes from client to server only.
FILE_PIPE_OUTBOUND 0x00000001	If this value is specified, the flow of data in the pipe goes from server to client only.
FILE_PIPE_FULL_DUPLEX 0x00000002	If this value is specified, the pipe is bi-directional; both server and client processes can read from and write to the pipe.

MaximumInstances (4 bytes): A 32-bit unsigned integer that contains the maximum number of instances that can be created for this pipe.

CurrentInstances (4 bytes): A 32-bit unsigned integer that contains the number of current named pipe instances.

InboundQuota (4 bytes): A 32-bit unsigned integer that contains the inbound quota, in bytes, for the named pipe. The inbound quota is the size of the buffer reserved for inbound transfer of data on the pipe.

ReadDataAvailable (4 bytes): A 32-bit unsigned integer that contains the bytes of data available to be read from the named pipe.

OutboundQuota (4 bytes): A 32-bit unsigned integer that contains the outbound quota, in bytes, for the named pipe. The outbound quota is the size of the buffer reserved for outbound transfer of data on the pipe.

WriteQuotaAvailable (4 bytes): A 32-bit unsigned integer that contains the write quota, in bytes, for the named pipe. If the **NamedPipeEnd** field is set to FILE_PIPE_CLIENT_END, the **WriteQuotaAvailable** field is the remaining **InboundQuota** field available. If the

NamedPipeEnd field is set to FILE_PIPE_SERVER_END, the **WriteQuotaAvailable** field is the remaining **OutboundQuota** field available.

NamedPipeState (4 bytes): A 32-bit unsigned integer that contains the named pipe state that specifies the connection status for the named pipe. MUST be one of the following.

Value	Meaning
FILE_PIPE_DISCONNECTED_STATE 0x00000001	Named pipe is disconnected.
FILE_PIPE_LISTENING_STATE 0x00000002	Named pipe is waiting to establish a connection.
FILE_PIPE_CONNECTED_STATE 0x00000003	Named pipe is connected.
FILE_PIPE_CLOSING_STATE 0x00000004	Named pipe is in the process of being closed.

NamedPipeEnd (4 bytes): A 32-bit unsigned integer that contains the type of the named pipe end, which specifies whether this is the client or the server side of a named pipe. MUST be one of the following.

Value	Meaning
FILE_PIPE_CLIENT_END 0x00000000	This is the client end of a named pipe.
FILE_PIPE_SERVER_END 0x00000001	This is the server end of a named pipe.

This operation returns a status code, as specified in [MS-ERREF] section 2.3. The status code returned directly by the function that processes this file information class MUST be STATUS_SUCCESS or one of the following.

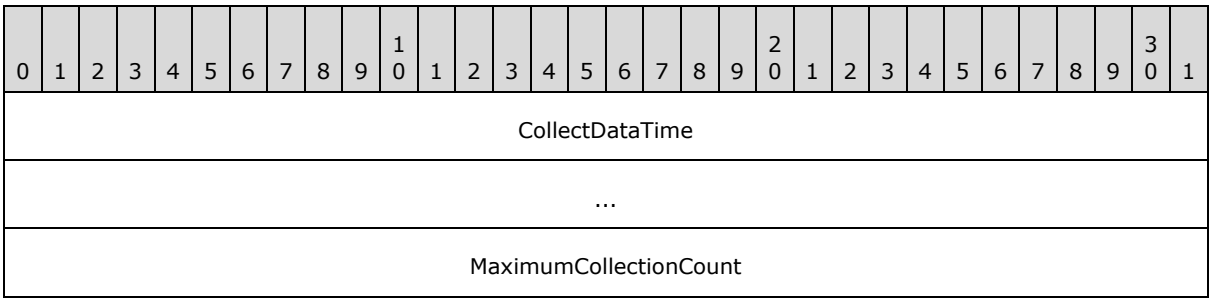
Error code	Meaning
STATUS_INFO_LENGTH_MISMATCH 0xC0000004	The specified information record length does not match the length that is required for the specified information class.

For more information on named pipes, please see [PIPE].

2.4.31 FilePipeRemoteInformation

This information class is used to query information on a named pipe that is associated with the client end of the pipe that is being queried. Remote information is not available for local pipes or for the server end of a remote pipe.

A **FILE_PIPE_REMOTE_INFORMATION** data element, defined as follows, is returned by the server.



CollectDataTime (8 bytes): A **LARGE_INTEGER** that MUST contain the maximum amount of time counted in 100-nanosecond intervals that will elapse before transmission of data from the client machine to the server.

MaximumCollectionCount (4 bytes): A 32-bit unsigned integer that MUST contain the maximum size, in bytes, of data that will be collected on the client machine before transmission to the server.

This operation returns a status code, as specified in [MS-ERREF] section 2.3. The status code returned directly by the function that processes this file information class MUST be STATUS_SUCCESS or one of the following.

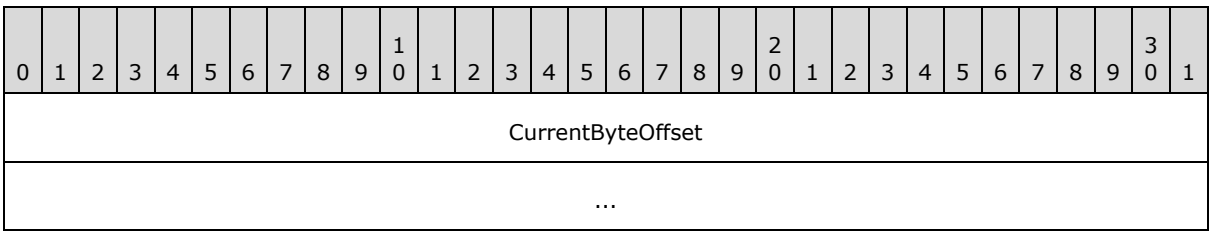
Error code	Meaning
STATUS_INFO_LENGTH_MISMATCH 0xC0000004	The specified information record length does not match the length that is required for the specified information class.

For more information on named pipes, please see [PIPE].

2.4.32 FilePositionInformation

This information class is used to query or set the position of the file pointer within a file.<97>

A **FILE_POSITION_INFORMATION** data element, defined as follows, is returned by the server or provided by the client.



CurrentByteOffset (8 bytes): A 64-bit signed integer that MUST contain the offset, in bytes, of the file pointer from the beginning of the file. A unique offset value is maintained for each open of a file. When setting the position, only values greater than or equal to zero are valid. If the given file was opened using the FILE_NO_INTERMEDIATE_BUFFERING flag, the offset that is being set SHOULD be aligned to a sector boundary. This value SHOULD<98> be updated by read and write operations if the given file was opened using the FILE_SYNCHRONOUS_IO_ALERT or FILE_SYNCHRONOUS_IO_NONALERT flags.

This operation returns a status code, as specified in [MS-ERREF] section 2.3. The status code returned directly by the function that processes this file information class MUST be STATUS_SUCCESS or one of the following.

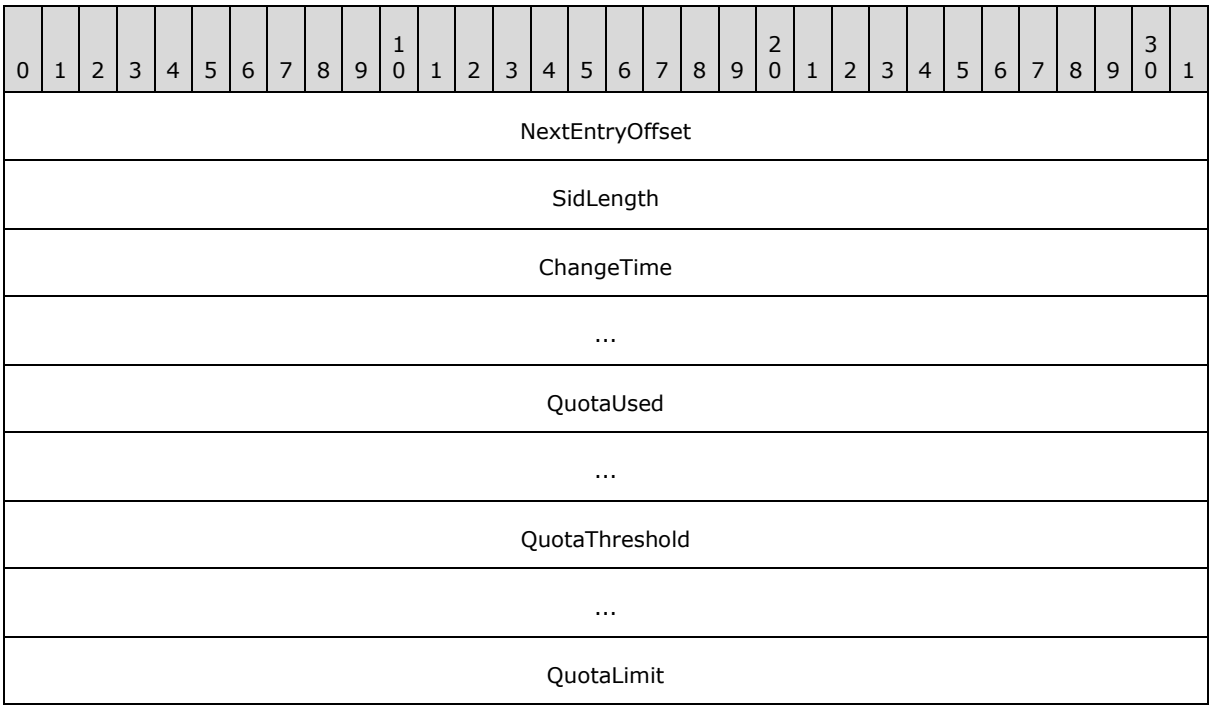
Error code	Meaning
STATUS_INFO_LENGTH_MISMATCH 0xC0000004	The specified information record length does not match the length that is required for the specified information class.
STATUS_INVALID_PARAMETER 0xC000000D	Returned when setting the offset if the **CurrentByteOffset** is negative or the file was opened using the FILE_NO_INTERMEDIATE_BUFFERING flag and **CurrentByteOffset** is not aligned to a sector boundary.

2.4.33 FileQuotaInformation

This information class is used to query or to set file quota information for a volume. For queries, an optional buffer of FILE_GET_QUOTA_INFORMATION (section 2.4.33.1) data elements is provided by the client to specify the SIDs for which quota information is requested. If the **FILE_GET_QUOTA_INFORMATION** buffer is not specified, information for all quotas is returned. A buffer of **FILE_QUOTA_INFORMATION** data elements is returned by the server. For sets, **FILE_QUOTA_INFORMATION** data elements are populated and sent by the client, as specified in [MS-SMB] section 2.2.7.4.1 and [MS-SMB2] section 3.2.4.15.<99>

When multiple **FILE_QUOTA_INFORMATION** data elements are present in the buffer, each MUST be aligned on an 8-byte boundary. Any bytes inserted for alignment SHOULD be set to zero, and the receiver MUST ignore them. No padding is required following the last data element.

A **FILE_QUOTA_INFORMATION** data element is as follows.



...
Sid (variable)
...

NextEntryOffset (4 bytes): A 32-bit unsigned integer that contains the byte offset from the beginning of this entry, at which the next FILE_QUOTA_INFORMATION entry is located, if multiple entries are present in a buffer. This member MUST be zero if no other entries follow this one. An implementation MUST use this value to determine the location of the next entry (if multiple entries are present in a buffer).

SidLength (4 bytes): A 32-bit unsigned integer that contains the length, in bytes, of the **Sid** data element.

ChangeTime (8 bytes): The last time that the quota was changed; see section 2.1.1. This value MUST be greater than or equal to 0x0000000000000000. When setting quota information, the server MUST ignore the value of this field.

QuotaUsed (8 bytes): A 64-bit signed integer that contains the amount of quota used by this user, in bytes. This value MUST be greater than or equal to 0x0000000000000000. When setting quota information, the server MUST ignore the value of this field.

QuotaThreshold (8 bytes): A 64-bit signed integer that contains the **disk quota** warning threshold, in bytes, on this volume for this user. This field MUST be set to a 64-bit integer value greater than or equal to 0 to set a quota warning threshold for this user on this volume. If this field is set to -1 there is no quota warning threshold for this user.

QuotaLimit (8 bytes): A 64-bit signed integer that contains the disk quota limit, in bytes, on this volume for this user. This field MUST be set to a 64-bit integer value greater than or equal to zero to set a disk quota limit for this user on this volume, to -1 to specify that no quota limit is set for this user, or to -2 to delete the quota entry for the user.

Sid (variable): Security identifier (SID) for this user.

This operation returns a status code, as specified in [MS-ERREF] section 2.3. The status code returned directly by the function that processes this file information class MUST be STATUS_SUCCESS or one of the following.

Error code	Meaning
STATUS_INVALID_DEVICE_REQUEST 0xC0000010	The target file system does not implement this functionality.
STATUS_INVALID_INFO_CLASS 0xC0000003	The specified information class is not a valid information class for the specified object.
STATUS_INFO_LENGTH_MISMATCH 0xC0000004	The specified information record length does not match the length that is required for the specified information class.
STATUS_INVALID_PARAMETER 0xC000000D	The SID or SID Length specified is not a valid parameter.
STATUS_NO_SUCH_FILE	For query operations, indicates that no

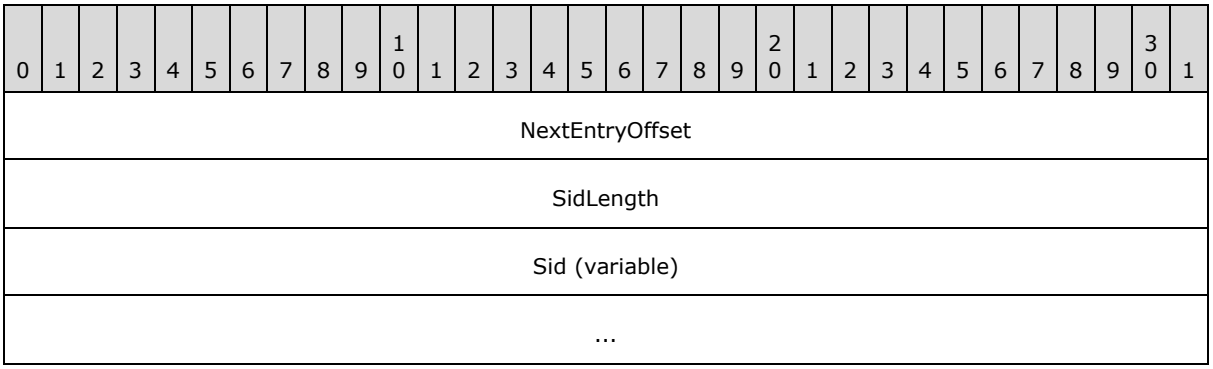
Error code	Meaning
0xC000000F	**FILE_QUOTA_INFORMATION** data elements were returned that matched the input criteria.
STATUS_BUFFER_TOO_SMALL 0xC0000023	The buffer is too small to contain the entry. No information has been written to the buffer.

2.4.33.1 FILE_GET_QUOTA_INFORMATION

This structure is used to provide the list of SIDs for which quota query information is requested.

When multiple FILE_GET_QUOTA_INFORMATION data elements are present in the buffer, each MUST be aligned on a 4-byte boundary. Any bytes inserted for alignment SHOULD be set to zero, and the receiver MUST ignore them. No padding is required following the last data element.

A **FILE_GET_QUOTA_INFORMATION** data element is as follows.



NextEntryOffset (4 bytes): A 32-bit unsigned integer that contains the byte offset from the beginning of this entry, at which the next FILE_GET_QUOTA_INFORMATION entry is located, if multiple entries are present in a buffer. This member MUST be zero if no other entries follow this one. An implementation MUST use this value to determine the location of the next entry (if multiple entries are present in a buffer).

SidLength (4 bytes): A 32-bit unsigned integer that contains the length, in bytes, of the **Sid** data element.

Sid (variable): SID for this user. SIDs are sent in little-endian format and require no padding. The format of a SID is as specified in [MS-DTYP] section 2.4.2.2.

2.4.34 FileRenameInformation

This information class is used to rename a file. The data element provided by the client takes one of two forms, depending on whether it is embedded within SMB or SMB2. The structure definitions are as follows:

- FILE_RENAME_INFORMATION_TYPE_1 for the SMB protocol (section 2.4.34.1).
- FILE_RENAME_INFORMATION_TYPE_2 for the SMB2 protocol (section 2.4.34.2).

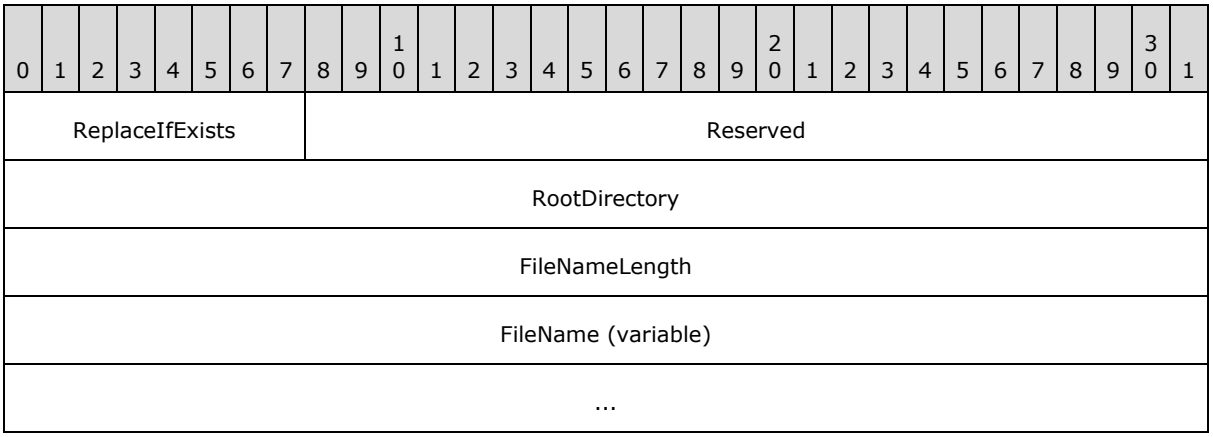
This operation returns a status code, as specified in [MS-ERREF] section 2.3. The status code returned directly by the function that processes this file information class MUST be STATUS_SUCCESS or one of the following.

Error code	Meaning
STATUS_INVALID_PARAMETER 0xC000000D	An invalid parameter was passed for **FileName** or **FileNameLength**, or the target file was open, or the **RootDirectory** field value was nonzero for a network operation.
STATUS_ACCESS_DENIED 0xC0000022	The handle was not opened with delete access.
STATUS_NOT_SAME_DEVICE 0xC00000D4	The destination file of a rename request is located on a different device than the source of the rename request.
STATUS_OBJECT_NAME_INVALID 0xC0000033	The object name is invalid for the target file system.
STATUS_OBJECT_NAME_COLLISION 0xC0000035	The specified name already exists and **ReplaceIfExists** is zero.
STATUS_INFO_LENGTH_MISMATCH 0xC0000004	The specified information record length does not match the length that is required for the specified information class.

2.4.34.1 FileRenameInformation for SMB

This information class is used to rename a file from within the SMB Protocol, as specified in [MS-SMB].

A **FILE_RENAME_INFORMATION_TYPE_1** data element, defined as follows, is provided by the client.



ReplaceIfExists (1 byte): A Boolean (section 2.1.8) value. Set to TRUE to indicate that if a file with the given name already exists, it SHOULD be replaced with the given file. Set to FALSE to indicate that the rename operation MUST fail if a file with the given name already exists.

Reserved (3 bytes): Reserved area for alignment. This field can contain any value and MUST be ignored.

RootDirectory (4 bytes): A 32-bit unsigned integer that contains the file handle for the directory to which the new name of the file is relative. For network operations, this value MUST be zero.

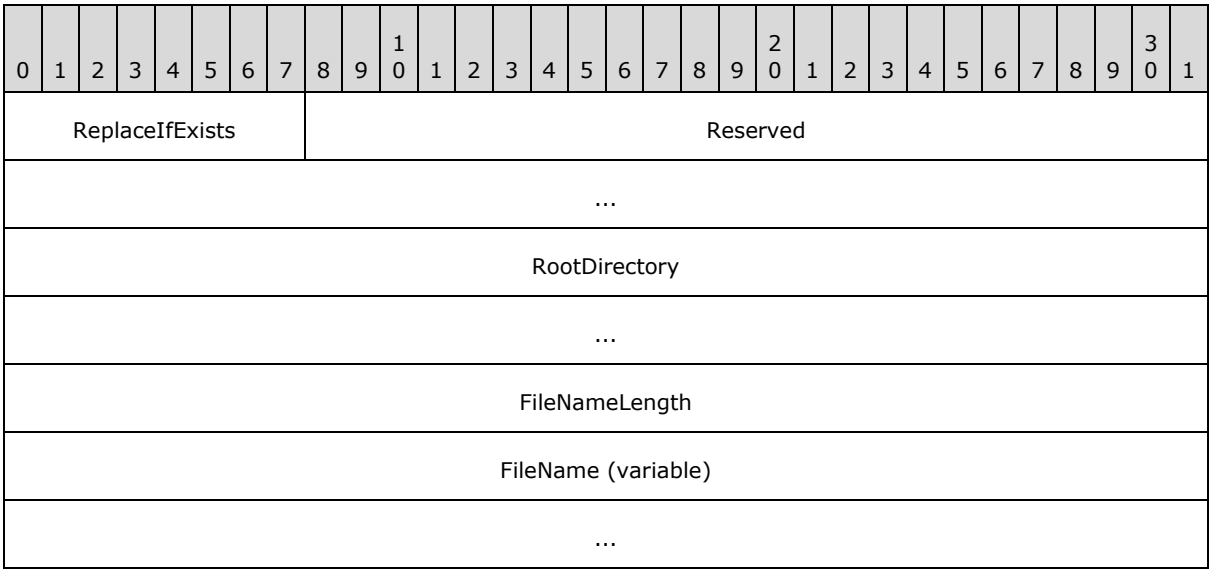
FileNameLength (4 bytes): A 32-bit unsigned integer that specifies the length, in bytes, of the file name contained within the **FileName** field.

FileName (variable): A sequence of Unicode characters containing the new name of the file. When working with this field, use **FileNameLength** to determine the length of the file name rather than assuming the presence of a trailing null delimiter. If the **RootDirectory** field is zero, this field MUST specify a full pathname to be assigned to the file. For network operations, this pathname is relative to the root of the share. If the **RootDirectory** field is not zero, this field MUST specify a pathname, relative to **RootDirectory**, for the new name of the file.

2.4.34.2 FileRenameInformation for SMB2

This information class is used to rename a file from within the SMB2 Protocol [MS-SMB2].

A **FILE_RENAME_INFORMATION_TYPE_2** data element, defined as follows, is provided by the client.



ReplaceIfExists (1 byte): A Boolean (section 2.1.8) value. Set to TRUE to indicate that if a file with the given name already exists, it SHOULD be replaced with the given file. Set to FALSE to indicate that the rename operation MUST fail if a file with the given name already exists.

Reserved (7 bytes): Reserved area for alignment. This field can contain any value and MUST be ignored.

RootDirectory (8 bytes): A 64-bit unsigned integer that contains the file handle for the directory to which the new name of the file is relative. For network operations, this value MUST always be zero.

FileNameLength (4 bytes): A 32-bit unsigned integer that specifies the length, in bytes, of the file name contained within the **FileName** field.

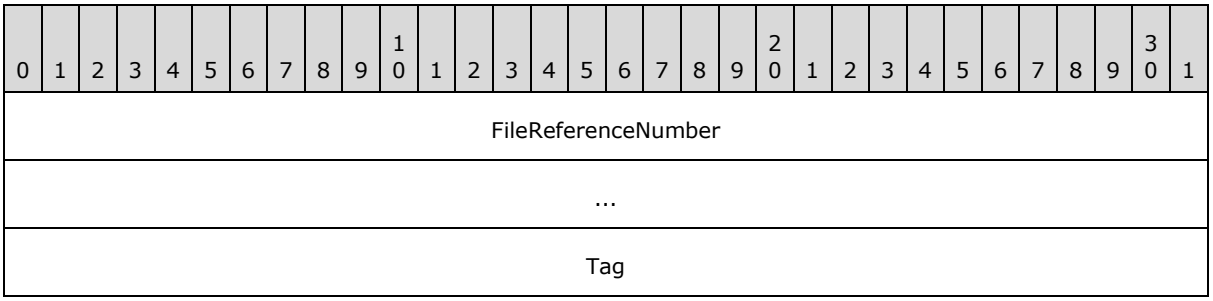
FileName (variable): A sequence of Unicode characters containing the new name of the file. When working with this field, use **FileNameLength** to determine the length of the file name rather than assuming the presence of a trailing null delimiter. If the **RootDirectory** field is zero, this member MUST specify a full pathname to be assigned to the file. For network

operations, this pathname is relative to the root of the share. If the **RootDirectory** field is not zero, this field MUST specify a pathname, relative to **RootDirectory**, for the new name of the file.

2.4.35 FileReparsePointInformation

This information class is used locally to query for information on a reparse point.

A **FILE_REPARSE_POINT_INFORMATION** data element, defined as follows, is returned to the caller.



FileReferenceNumber (8 bytes): A 64-bit signed integer that contains the file reference number for the file. NTFS generates this number and assigns it to the file automatically when the file is created. The value of this field MUST be greater than or equal to 0.

Tag (4 bytes): A 32-bit unsigned integer value containing the reparse point tag that uniquely identifies the owner of the reparse point. Section 2.1.2.1 contains more details on reparse tags.

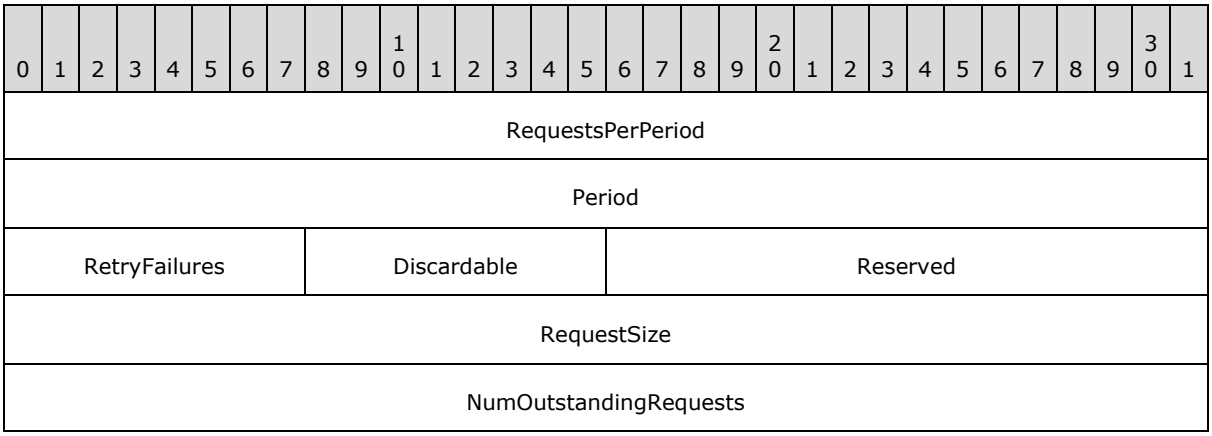
This operation returns a status code, as specified in [MS-ERREF] section 2.3. The status code returned directly by the function that processes this file information class MUST be STATUS_SUCCESS or one of the following.

Error code	Meaning
STATUS_INFO_LENGTH_MISMATCH 0xC0000004	The specified information record length does not match the length that is required for the specified information class.
STATUS_INVALID_DEVICE_REQUEST 0xC0000010	The target file system does not implement this functionality.
STATUS_INVALID_INFO_CLASS 0xC0000003	The specified information class is not a valid information class for the specified object.
STATUS_NO_SUCH_FILE 0xC000000F	No reparse points exist for the given file.
STATUS_BUFFER_OVERFLOW 0x80000005	The output buffer was filled before all of the FILE_REPARSE_POINT_INFORMATION structures could be returned; a partial structure might be returned.

2.4.36 FileSfioReserveInformation

This information class is used locally to query or set reserved bandwidth for a file handle. Conceptually reserving bandwidth is effectively specifying the bytes per second to allocate to file IO.

A **FILE_SFIO_RESERVE_INFORMATION** data element, defined as follows, is returned to the caller.



RequestsPerPeriod (4 bytes): A 32-bit unsigned integer indicating the number of I/O requests that must complete per period of time, as specified in the **Period** field. When setting bandwidth reservation, a value of 0 indicates to the file system that it MUST free any existing reserved bandwidth.

Period (4 bytes): A 32-bit unsigned integer that contains the period for reservation, which is the time from which I/O is issued to the kernel until the time the I/O should be completed, specified in milliseconds.

RetryFailures (1 byte): A Boolean (section 2.1.8) value.

Discardable (1 byte): A Boolean (section 2.1.8) value.

Reserved (2 bytes): Reserved for alignment. This field can contain any value and MUST be ignored.

RequestSize (4 bytes): A 32-bit unsigned integer that indicates the minimum size of any individual I/O request that may be issued by an application using bandwidth reservation. When setting reservations, this field MUST be ignored by servers and SHOULD be set to 0 by clients.

NumOutstandingRequests (4 bytes): A 32-bit unsigned integer that indicates the number of RequestSize I/O requests allowed to be outstanding at any time. When setting reservations, this field MUST be ignored by servers and SHOULD be set to 0 by clients.

This operation returns a status code, as specified in [MS-ERREF] section 2.3. The status code returned directly by the function that processes this file information class MUST be STATUS_SUCCESS or one of the following.

Error code	Meaning
STATUS_NOT_SUPPORTED	The request is not supported.

Error code	Meaning
0xC00000BB	
STATUS_INFO_LENGTH_MISMATCH 0xC0000004	The specified information record length does not match the length that is required for the specified information class.

2.4.37 FileShortNameInformation

This information class is used to change a file's short name. If the supplied name is of zero length, the file's existing short name, if any, SHOULD<100> be deleted. Otherwise, the supplied name MUST be a valid short name as specified in section 2.1.5.2.1, and must be unique among all file names and short names in the same directory as the file being operated on. A caller changing the file's short name MUST have SeRestorePrivilege, as specified in [MS-LSAD] section 3.1.1.2.1.

A FILE_NAME_INFORMATION (section 2.1.7) data element is provided by the client.

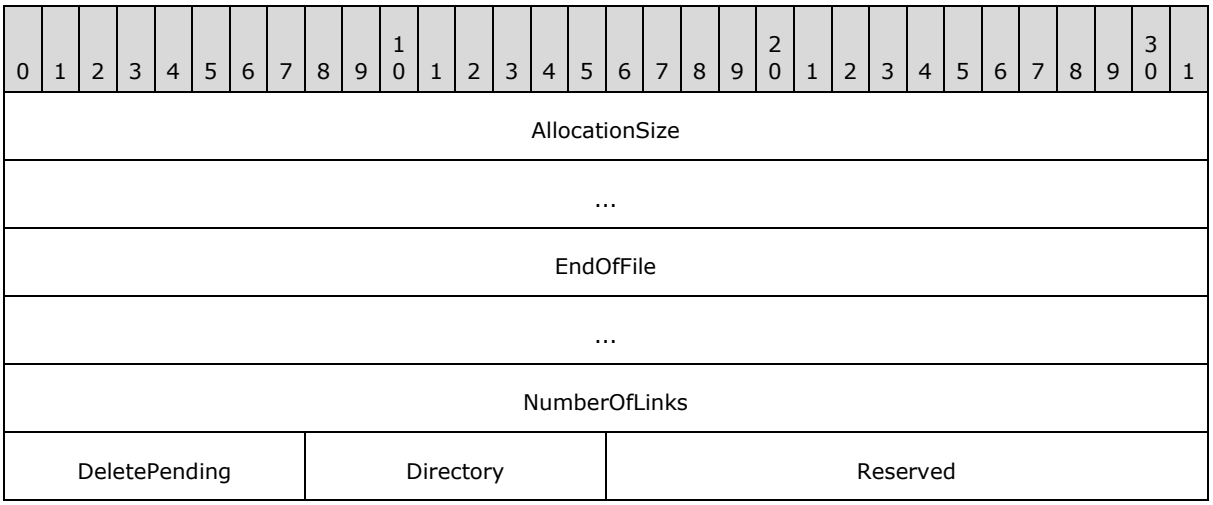
This operation returns a status code, as specified in [MS-ERREF] section 2.3. The status code returned directly by the function that processes this file information class MUST be STATUS_SUCCESS or one of the following.

Error code	Meaning
STATUS_MEDIA_WRITE_PROTECTED 0xC00000A2	The target cannot be written to because it is write-protected.
STATUS_INVALID_PARAMETER 0xC000000D	The file name is not a valid parameter.
STATUS_ACCESS_DENIED 0xC0000022	The handle was not opened to write file data or file attributes, or the file has been deleted.
STATUS_PRIVILEGE_NOT_HELD 0xC0000061	The SeRestorePrivilege privilege is not held.
STATUS_SHORT_NAMES_NOT_ENABLED_ON_VOLUME 0xC000019F	Short names are not enabled on this volume.
STATUS_OBJECT_NAME_COLLISION 0xC0000035	The specified name already exists.
STATUS_INFO_LENGTH_MISMATCH 0xC0000004	The specified information record length does not match the length that is required for the specified information class.

2.4.38 FileStandardInformation

This information class is used to query file information.

A **FILE_STANDARD_INFORMATION** data element, defined as follows, is returned by the server.



AllocationSize (8 bytes): A 64-bit signed integer that contains the file allocation size, in bytes. The value of this field MUST be an integer multiple of the cluster size.

EndOfFile (8 bytes): A 64-bit signed integer that contains the absolute end-of-file position as a byte offset from the start of the file. **EndOfFile** specifies the offset to the byte immediately following the last valid byte in the file. Because this value is zero-based, it actually refers to the first free byte in the file. That is, it is the offset from the beginning of the file at which new bytes appended to the file will be written. The value of this field MUST be greater than or equal to 0.

NumberOfLinks (4 bytes): A 32-bit unsigned integer that contains the number of non-deleted links to this file.

DeletePending (1 byte): A Boolean (section 2.1.8) value. Set to TRUE to indicate that a file deletion has been requested; set to FALSE otherwise.

Directory (1 byte): A Boolean (section 2.1.8) value. Set to TRUE to indicate that the file is a directory; set to FALSE otherwise.

Reserved (2 bytes): A 16-bit field. This field is reserved. This field can be set to any value, and MUST be ignored.

This operation returns a status code, as specified in [MS-ERREF] section 2.3. The status code returned directly by the function that processes this file information class MUST be STATUS_SUCCESS or one of the following.

Error code	Meaning
STATUS_INFO_LENGTH_MISMATCH 0xC0000004	The specified information record length does not match the length that is required for the specified information class.

2.4.39 FileStandardLinkInformation

This information class is used locally to query file link information.<101>

A **FILE_STANDARD_LINK_INFORMATION** data element, defined as follows, is returned to the caller.

0	1	2	3	4	5	6	7	8	9	10	1	2	3	4	5	6	7	8	9	20	1	2	3	4	5	6	7	8	9	30	1
NumberOfAccessibleLinks																															
TotalNumberOfLinks																															
DeletePending								Directory								Reserved															

NumberOfAccessibleLinks (4 bytes): A 32-bit unsigned integer that contains the number of non-deleted links to this file.

TotalNumberOfLinks (4 bytes): A 32-bit unsigned integer that contains the total number of links to this file, including links marked for delete.

DeletePending (1 byte): A Boolean (section 2.1.8) value that MUST be set to TRUE to indicate that a file deletion has been requested; otherwise, FALSE.

Directory (1 byte): An 8-bit field that MUST be set to 1 to indicate that the file is a directory; otherwise, 0.

Reserved (2 bytes): A 16-bit field. This field is reserved. This field can be set to any value and MUST be ignored.

This operation returns a status code, as specified in [MS-ERREF] section 2.3. The status code returned directly by the function that processes this file information class MUST be STATUS_SUCCESS or one of the following.

Error code	Meaning
STATUS_STATUS_NOT_SUPPORTED 0xC00000BB	The request is not supported.
STATUS_INFO_LENGTH_MISMATCH 0xC0000004	The specified information record length does not match the length that is required for the specified information class.

2.4.40 FileStreamInformation

This information class is used to enumerate the data streams of a file or a directory. A buffer of **FILE_STREAM_INFORMATION** data elements is returned by the server.

When multiple **FILE_STREAM_INFORMATION** data elements are present in the buffer, each MUST be aligned on an 8-byte boundary; any bytes inserted for alignment SHOULD be set to zero and the receiver MUST ignore them. No padding is required following the last data element.

A **FILE_STREAM_INFORMATION** data element is as follows.

0	1	2	3	4	5	6	7	8	9	10	1	2	3	4	5	6	7	8	9	20	1	2	3	4	5	6	7	8	9	30	1
NextEntryOffset																															

StreamNameLength
StreamSize
...
StreamAllocationSize
...
StreamName (variable)
...

NextEntryOffset (4 bytes): A 32-bit unsigned integer that contains the byte offset from the beginning of this entry, at which the next **FILE_STREAM_INFORMATION** entry is located, if multiple entries are present in a buffer. This member is zero if no other entries follow this one. An implementation MUST use this value to determine the location of the next entry (if multiple entries are present in a buffer).

StreamNameLength (4 bytes): A 32-bit unsigned integer that contains the length, in bytes, of the stream name string.

StreamSize (8 bytes): A 64-bit signed integer that contains the size, in bytes, of the stream. The value of this field MUST be greater than or equal to 0x0000000000000000.

StreamAllocationSize (8 bytes): A 64-bit signed integer that contains the file stream allocation size, in bytes. The value of this field MUST be an integer multiple of the cluster size.

StreamName (variable): A sequence of Unicode characters containing the name of the stream using the form ":streamname:$DATA", or "::$DATA" for the default data stream, as specified in section 2.1.4. This field is not null-terminated and MUST be handled as a sequence of **StreamNameLength** bytes.

This operation returns a status code, as specified in [MS-ERREF] section 2.3. The status code returned directly by the function that processes this file information class MUST be STATUS_SUCCESS or one of the following.

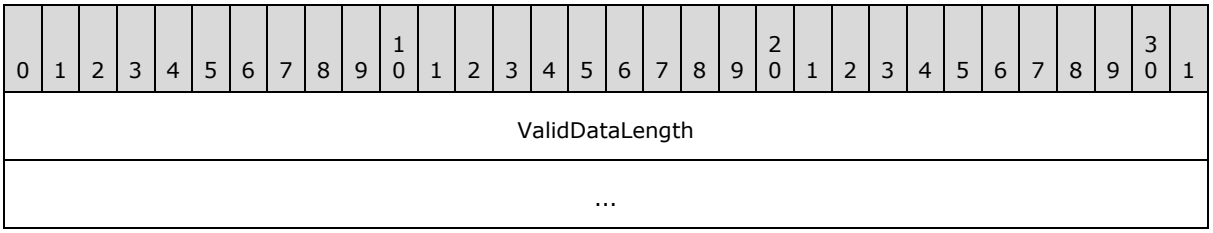
Error code	Meaning
STATUS_BUFFER_OVERFLOW 0x80000005	The output buffer was filled before all of the stream information could be returned. Only complete FILE_STREAM_INFORMATION structures are returned.
STATUS_INFO_LENGTH_MISMATCH 0xC0000004	The specified information record length does not match the length that is required for the specified information class.

2.4.41 FileValidDataLengthInformation

This information class is used to set the valid data length information for a file. A file's valid data length is the length, in bytes, of the data that has been written to the file. This valid data extends

from the beginning of the file to the last byte in the file that has not been zeroed or left uninitialized.<102>

A **FILE_VALID_DATA_LENGTH_INFORMATION** data element, defined as follows, is provided by the client.



ValidDataLength (8 bytes): A 64-bit signed integer that contains the new valid data length for the file. This parameter MUST be a positive value that is greater than the current valid data length, but less than or equal to the current file size.

This operation returns a status code, as specified in [MS-ERREF] section 2.3. The status code returned directly by the function that processes this file information class MUST be STATUS_SUCCESS or one of the following.

Error code	Meaning
STATUS_MEDIA_WRITE_PROTECTED 0xC00000A2	The target cannot be written to because it is write-protected.
STATUS_INVALID_PARAMETER 0xC000000D	The *ValidDataLength* specified is not a valid parameter or the given handle is to a sparse or compressed file.
STATUS_PRIVILEGE_NOT_HELD 0xC0000061	The manage volume privilege is not held.
STATUS_INFO_LENGTH_MISMATCH 0xC0000004	The specified information record length does not match the length that is required for the specified information class.

2.5 File System Information Classes

File system information classes are numerical values (specified by the Level column in the following table) that specify what information on a particular instance of a file system on a volume is to be queried. File system information classes can retrieve information such as the file system type, volume label, size of the file system, and name of the driver used to access the file system. The table indicates which file system information classes are supported for query and set operations.<103>

File system information class	Level	Uses
FileFsVolumeInformation	1	Query
FileFsLabelInformation	2	LOCAL<104>
FileFsSizeInformation	3	Query
FileFsDeviceInformation	4	Query

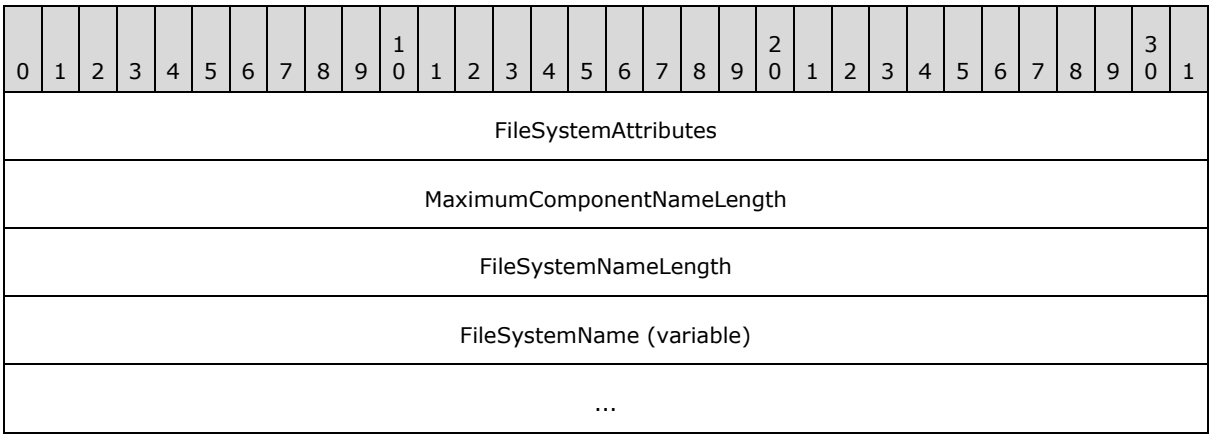
File system information class	Level	Uses
FileFsAttributeInformation	5	Query
FileFsControlInformation	6	Query, Set
FileFsFullSizeInformation	7	Query
FileFsObjectIdInformation	8	Query, Set
FileFsDriverPathInformation	9	LOCAL<105>
FileFsVolumeFlagsInformation	10	LOCAL<106>

If an Information Class is specified that does not match the usage in the above table, STATUS_INVALID_INFO_CLASS MUST be returned. If a file system does not implement one of the above defined uses of an Information Class, STATUS_INVALID_PARAMETER MUST be returned.

2.5.1 FileFsAttributeInformation

This information class is used to query attribute information for a file system.

A **FILE_FS_ATTRIBUTE_INFORMATION** data element, defined as follows, is returned by the server.



FileSystemAttributes (4 bytes): A 32-bit unsigned integer that contains a bitmask of flags that specify attributes of the specified file system as a combination of the following flags. The value of this field MUST be a bitwise OR of zero or more of the following with the exception that FILE_FILE_COMPRESSION and FILE_VOLUME_IS_COMPRESSED cannot both be set. Any flag values not explicitly mentioned here can be set to any value, and MUST be ignored.

Value	Meaning
FILE_SUPPORTS_USN_JOURNAL 0x02000000	The file system implements a USN change journal<107>.
FILE_SUPPORTS_OPEN_BY_FILE_ID 0x01000000	The file system supports opening a file by FileID or ObjectID<108>.
FILE_SUPPORTS_EXTENDED_ATTRIBUTES	The file system persistently stores Extended Attribute

Value	Meaning
0x00800000	information per file<109>.
FILE_SUPPORTS_HARD_LINKS 0x00400000	The file system supports hard linking files<110>.
FILE_SUPPORTS_TRANSACTIONS 0x00200000	The volume supports transactions. <111>
FILE_SEQUENTIAL_WRITE_ONCE 0x00100000	The underlying volume is write once.
FILE_READ_ONLY_VOLUME 0x00080000	If set, the volume has been mounted in read-only mode<112>
FILE_NAMED_STREAMS 0x00040000	The file system supports **named streams**.
FILE_SUPPORTS_ENCRYPTION 0x00020000	The file system supports the Encrypted File System (EFS). <113>
FILE_SUPPORTS_OBJECT_IDS 0x00010000	The file system supports object identifiers.
FILE_VOLUME_IS_COMPRESSED 0x00008000	The specified volume is a compressed volume. This flag is incompatible with the FILE_FILE_COMPRESSION flag.
FILE_SUPPORTS_REMOTE_STORAGE 0x00000100	The file system supports remote storage. <114>
FILE_SUPPORTS_REPARSE_POINTS 0x00000080	The file system supports reparse points.
FILE_SUPPORTS_SPARSE_FILES 0x00000040	The file system supports sparse files.
FILE_VOLUME_QUOTAS 0x00000020	The file system supports per-user quotas.
FILE_FILE_COMPRESSION 0x00000010	The file volume supports file-based compression. This flag is incompatible with the FILE_VOLUME_IS_COMPRESSED flag.
FILE_PERSISTENT_ACLS 0x00000008	The file system preserves and enforces access control lists (ACLs).
FILE_UNICODE_ON_DISK 0x00000004	The file system supports Unicode in file and directory names. This flag applies only to file and directory names; the file system neither restricts nor interprets the bytes of data within a file.
FILE_CASE_PRESERVED_NAMES 0x00000002	The file system preserves the case of file names when it places a name on disk.
FILE_CASE_SENSITIVE_SEARCH 0x00000001	The file system supports case-sensitive file names when looking up (searching for) file names in a directory.

MaximumComponentNameLength (4 bytes): A 32-bit signed integer that contains the maximum **file name component** length, in bytes, supported by the specified file system. The value of this field MUST be greater than zero and MUST be no more than 510.<115>

FileSystemNameLength (4 bytes): A 32-bit unsigned integer that contains the length, in bytes, of the file system name in the **FileSystemName** field. The value of this field MUST be greater than 0.

FileSystemName (variable): A variable-length Unicode field containing the name of the file system. This field is not null-terminated and MUST be handled as a sequence of **FileSystemNameLength** bytes. This field is intended to be informative only. A client SHOULD NOT infer file system type specific behavior from this field. <116>

This operation returns a status code, as specified in [MS-ERREF] section 2.3. The status code returned directly by the function that processes this file information class MUST be STATUS_SUCCESS or one of the following.

Error code	Meaning
STATUS_INFO_LENGTH_MISMATCH 0xC0000004	The specified information record length does not match the length that is required for the specified information class.
STATUS_BUFFER_OVERFLOW 0x80000005	The output buffer was filled before all of the file system information could be returned; only a portion of the FileSystemName field is returned.

2.5.2 FileFsControlInformation

This information class is used to query or set quota and content indexing control information for a file system volume.

Setting quota information requires the caller to have permission to open a volume handle or a handle to the quota index file for write access.

A **FILE_FS_CONTROL_INFORMATION** data element, defined as follows, is returned by the server or provided by the client.

0	1	2	3	4	5	6	7	8	9	10	1	2	3	4	5	6	7	8	9	20	1	2	3	4	5	6	7	8	9	30	1
FreeSpaceStartFiltering																															
...																															
FreeSpaceThreshold																															
...																															
FreeSpaceStopFiltering																															
...																															

DefaultQuotaThreshold
...
DefaultQuotaLimit
...
FileSystemControlFlags
Padding

FreeSpaceStartFiltering (8 bytes): A 64-bit signed integer that contains the minimum amount of free disk space, in bytes, that is required for the operating system's **content indexing service** to begin document filtering. This value SHOULD be set to 0, and MUST be ignored.<117>

FreeSpaceThreshold (8 bytes): A 64-bit signed integer that contains the minimum amount of free disk space, in bytes, that is required for the indexing service to continue to filter documents and merge word lists. This value SHOULD be set to 0, and MUST be ignored.<118>

FreeSpaceStopFiltering (8 bytes): A 64-bit signed integer that contains the minimum amount of free disk space, in bytes, that is required for the content indexing service to continue filtering. This value SHOULD be set to 0, and MUST be ignored.<119>

DefaultQuotaThreshold (8 bytes): A 64-bit unsigned integer that contains the default per-user disk quota warning threshold, in bytes, for the volume. A value of 0xFFFFFFFFFFFFFFFF specifies that no default quota warning threshold per user is set.

DefaultQuotaLimit (8 bytes): A 64-bit unsigned integer that contains the default per-user disk quota limit, in bytes, for the volume. A value of 0xFFFFFFFFFFFFFFFF specifies that no default quota limit per user is set.

FileSystemControlFlags (4 bytes): A 32-bit unsigned integer that contains a bitmask of flags that control quota enforcement and logging of user-related quota events on the volume. The following bit flags are valid in any combination. Bits not defined in the following table SHOULD be set to 0, and MUST be ignored.<120>

Value	Meaning
FILE_VC_CONTENT_INDEX_DISABLED 0x00000008	Content indexing is disabled.
FILE_VC_LOG_QUOTA_LIMIT 0x00000020	An event log entry will be created when the user exceeds the assigned disk quota limit.
FILE_VC_LOG_QUOTA_THRESHOLD 0x00000010	An event log entry will be created when the user exceeds his or her assigned quota warning threshold.
FILE_VC_LOG_VOLUME_LIMIT 0x00000080	An event log entry will be created when the volume's free space limit is exceeded.

Value	Meaning
FILE_VC_LOG_VOLUME_THRESHOLD 0x00000040	An event log entry will be created when the volume's free space threshold is exceeded.
FILE_VC_QUOTA_ENFORCE 0x00000002	Quotas are tracked and enforced on the volume. Note: FILE_VC_QUOTA_TRACK takes precedence over this flag. In other words, if both FILE_VC_QUOTA_TRACK and FILE_VC_QUOTA_ENFORCE are set, the FILE_VC_QUOTA_ENFORCE flag is ignored. This flag will be ignored if a client attempts to set it.
FILE_VC_QUOTA_TRACK 0x00000001	Quotas are tracked on the volume, but they are not enforced. Tracked quotas enable reporting on the file system space used by system users. If both this flag and FILE_VC_QUOTA_ENFORCE are specified, FILE_VC_QUOTA_ENFORCE is ignored. Note: This flag takes precedence over FILE_VC_QUOTA_ENFORCE. In other words, if both FILE_VC_QUOTA_TRACK and FILE_VC_QUOTA_ENFORCE are set, the FILE_VC_QUOTA_ENFORCE flag is ignored. This flag will be ignored if a client attempts to set it.
FILE_VC_QUOTAS_INCOMPLETE 0x00000100	The quota information for the volume is incomplete because it is corrupt, or the system is in the process of rebuilding the quota information. Note: This does not necessarily imply that FILE_VC_QUOTAS_REBUILDING is set. This flag will be ignored if a client attempts to set it.
FILE_VC_QUOTAS_REBUILDING 0x00000200	The file system is rebuilding the quota information for the volume. Note: This does not necessarily imply that FILE_VC_QUOTAS_INCOMPLETE is set. This flag will be ignored if a client attempts to set it.

Padding (4 bytes): This field SHOULD be set to 0x00000000 and MUST be ignored.

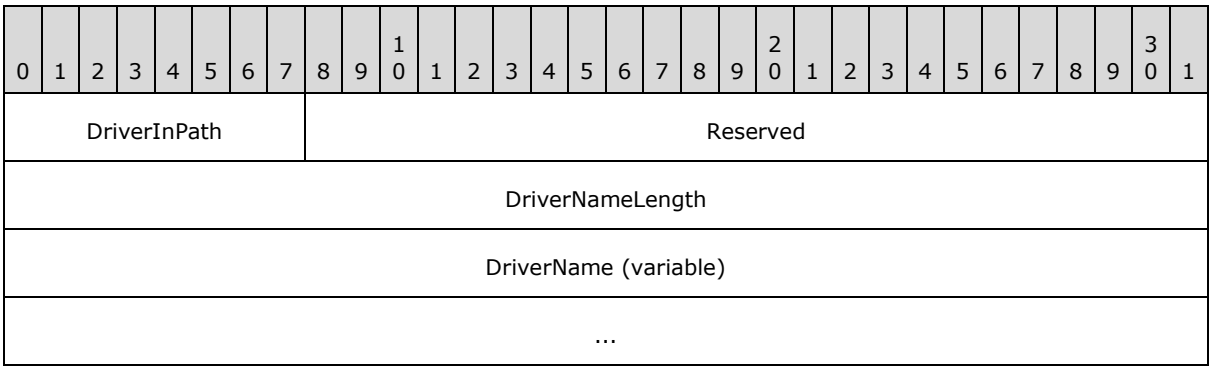
This operation returns a status code, as specified in [MS-ERREF] section 2.3. The status code returned directly by the function that processes this file information class MUST be STATUS_SUCCESS or one of the following.

Error code	Meaning
STATUS_INFO_LENGTH_MISMATCH 0xC0000004	The specified information record length does not match the length that is required for the specified information class.
STATUS_VOLUME_NOT_UPGRADED 0xC000029C	The file system on the volume does not support quotas.

2.5.3 FileFsDriverPathInformation

This information class is used locally to query if a given driver is in the I/O path for a file system volume.

A **FILE_FS_DRIVER_PATH_INFORMATION** data element, defined as follows, is returned to the caller.



DriverInPath (1 byte): A Boolean (section 2.1.8) value. Set to TRUE if the driver is in the I/O path for the file system volume; set to FALSE otherwise.

Reserved (3 bytes): Reserved for alignment. This field can contain any value and MUST be ignored.

DriverNameLength (4 bytes): A 32-bit unsigned integer that contains the length of the **DriverName** string.

DriverName (variable): A variable-length Unicode field containing the name of the driver for which to query. This sequence of Unicode characters MUST NOT be null-terminated.

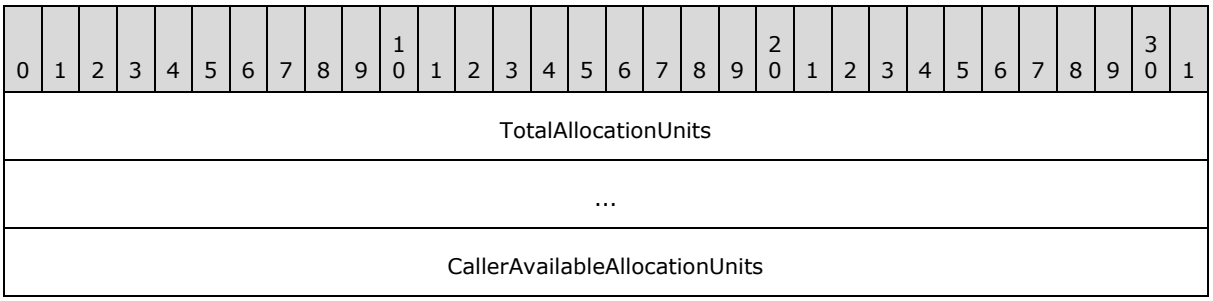
This operation returns a status code, as specified in [MS-ERREF] section 2.3. The status code returned directly by the function that processes this file information class MUST be STATUS_SUCCESS or one of the following.

Error code	Meaning
STATUS_INFO_LENGTH_MISMATCH 0xC0000004	The specified information record length does not match the length that is required for the specified information class.

2.5.4 FileFsFullSizeInformation

This information class is used to query sector size information for a file system volume.

A **FILE_FS_FULL_SIZE_INFORMATION** data element, defined as follows, is returned by the server.



...
ActualAvailableAllocationUnits
...
SectorsPerAllocationUnit
BytesPerSector

TotalAllocationUnits (8 bytes): A 64-bit signed integer that contains the total number of allocation units on the volume that are available to the user associated with the calling thread. The value of this field MUST be greater than or equal to 0.<121>

CallerAvailableAllocationUnits (8 bytes): A 64-bit signed integer that contains the total number of free allocation units on the volume that are available to the user associated with the calling thread. The value of this field MUST be greater than or equal to 0.<122>

ActualAvailableAllocationUnits (8 bytes): A 64-bit signed integer that contains the total number of free allocation units on the volume. The value of this field MUST be greater than or equal to 0.

SectorsPerAllocationUnit (4 bytes): A 32-bit unsigned integer that contains the number of sectors in each allocation unit.

BytesPerSector (4 bytes): A 32-bit unsigned integer that contains the number of bytes in each sector.

This operation returns a status code, as specified in [MS-ERREF] section 2.3. The status code returned directly by the function that processes this file information class MUST be STATUS_SUCCESS or one of the following.

Error code	Meaning
STATUS_INFO_LENGTH_MISMATCH 0xC0000004	The specified information record length does not match the length that is required for the specified information class.

2.5.5 FileFsLabelInformation

This information class is used locally to set the label for a file system volume.

A **FILE_FS_LABEL_INFORMATION** data element, defined as follows, is provided by the caller.

0	1	2	3	4	5	6	7	8	9	10	1	2	3	4	5	6	7	8	9	20	1	2	3	4	5	6	7	8	9	30	1
VolumeLabelLength																															
VolumeLabel (variable)																															
...																															

VolumeLabelLength (4 bytes): A 32-bit unsigned integer that contains the length, in bytes, including the trailing null, if present, of the name for the volume.<123>

VolumeLabel (variable): A variable-length Unicode field containing the name of the volume. The content of this field can be a null-terminated string, or it can be a string padded with the space character to be **VolumeLabelLength** bytes long.

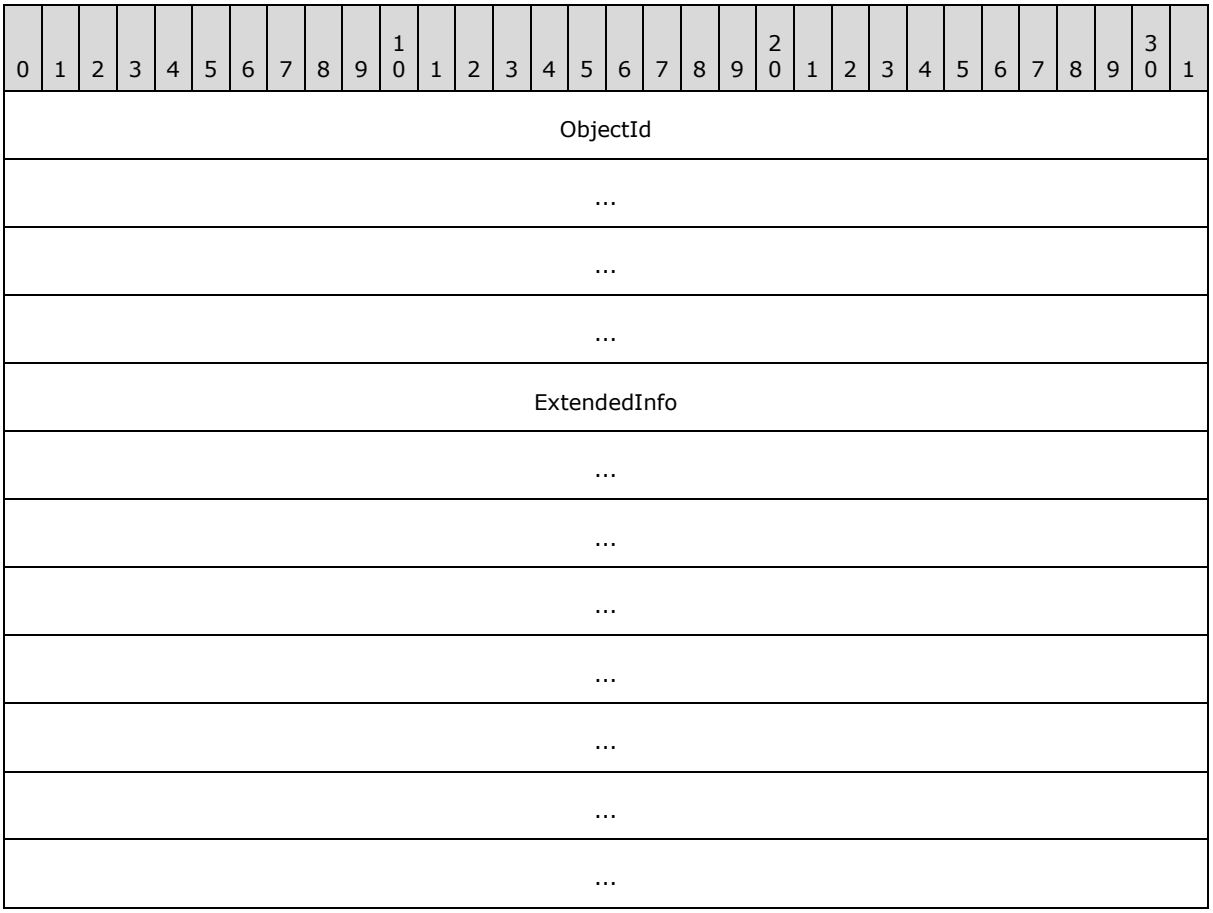
This operation returns a status code, as specified in [MS-ERREF] section 2.3. The status code returned directly by the function that processes this file information class MUST be STATUS_SUCCESS or one of the following.

Error code	Meaning
STATUS_INFO_LENGTH_MISMATCH 0xC0000004	The specified information record length does not match the length that is required for the specified information class.

2.5.6 FileFsObjectIdInformation

This information class is used to query or set the object ID for a file system data element. The operation MUST fail if the file system does not support object IDs. <124>

A **FILE_FS_OBJECTID_INFORMATION** data element, defined as follows, is returned by the server or provided by the client.



(ExtendedInfo cont'd for 4 rows)

ObjectId (16 bytes): A 16-byte GUID that identifies the file system volume on the disk. This value is not required to be unique on the system.

ExtendedInfo (48 bytes): A 48-byte value containing extended information on the file system volume. If no extended information has been written for this file system volume, the server MUST return 48 bytes of 0x00 in this field.<125>

This operation returns a status code, as specified in [MS-ERREF] section 2.3. The status code returned directly by the function that processes this file information class MUST be STATUS_SUCCESS or one of the following.

Error code	Meaning
STATUS_INFO_LENGTH_MISMATCH 0xC0000004	The specified information record length does not match the length that is required for the specified information class.
STATUS_VOLUME_NOT_UPGRADED 0xC000029C	The file system on the volume does not support object IDs.

2.5.7 FileFsSizeInformation

This information class is used to query sector size information for a file system volume.

A **FILE_FS_SIZE_INFORMATION** data element, defined as follows, is returned by the server.

0	1	2	3	4	5	6	7	8	9	10	1	2	3	4	5	6	7	8	9	20	1	2	3	4	5	6	7	8	9	30	1
TotalAllocationUnits																															
...																															
AvailableAllocationUnits																															
...																															
SectorsPerAllocationUnit																															
BytesPerSector																															

TotalAllocationUnits (8 bytes): A 64-bit signed integer that contains the total number of allocation units on the volume that are available to the user associated with the calling thread. This value MUST be greater than or equal to 0.<126>

AvailableAllocationUnits (8 bytes): A 64-bit signed integer that contains the total number of free allocation units on the volume that are available to the user associated with the calling thread. This value MUST be greater than or equal to 0.<127>

SectorsPerAllocationUnit (4 bytes): A 32-bit unsigned integer that contains the number of sectors in each allocation unit.

BytesPerSector (4 bytes): A 32-bit unsigned integer that contains the number of bytes in each sector.

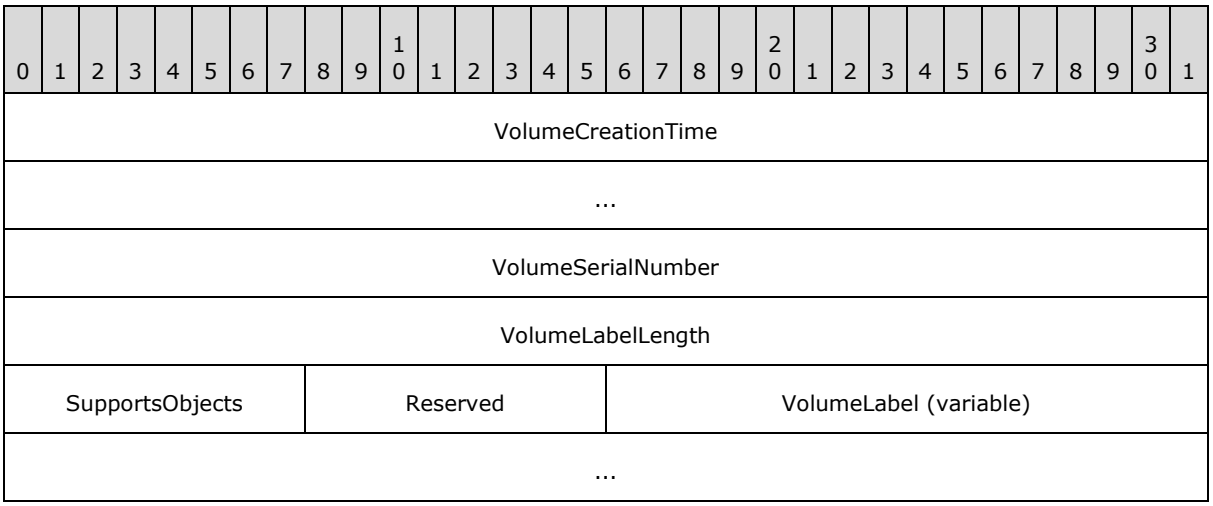
This operation returns a status code, as specified in [MS-ERREF] section 2.3. The status code returned directly by the function that processes this file information class MUST be STATUS_SUCCESS or one of the following.

Error code	Meaning
STATUS_INFO_LENGTH_MISMATCH 0xC0000004	The specified information record length does not match the length that is required for the specified information class.

2.5.8 FileFsVolumeInformation

This information class is used to query information on a volume on which a file system is mounted.

A **FILE_FS_VOLUME_INFORMATION** data element, defined as follows, is returned by the server.



VolumeCreationTime (8 bytes): The time when the volume was created; see section 2.1.1. The value of this field MUST be greater than or equal to 0.

VolumeSerialNumber (4 bytes): A 32-bit unsigned integer that contains the serial number of the volume. The serial number is an opaque value generated by the file system at format time, and is not necessarily related to any hardware serial number for the device on which the file system is located. No specific format or content of this field is required for protocol interoperation. This value is not required to be unique.

VolumeLabelLength (4 bytes): A 32-bit unsigned integer that contains the length, in bytes, including the trailing null, if present, of the name of the volume<128>

SupportsObjects (1 byte): A Boolean (section 2.1.8) value. Set to TRUE if the file system supports **object-oriented file system** objects; set to FALSE otherwise.<129>

Reserved (1 byte): An 8-bit field. This field is reserved. This field MUST be set to zero and MUST be ignored.

VolumeLabel (variable): A variable-length Unicode field containing the name of the volume. The content of this field can be a null-terminated string or can be a string padded with the space character to be **VolumeLabelLength** bytes long.

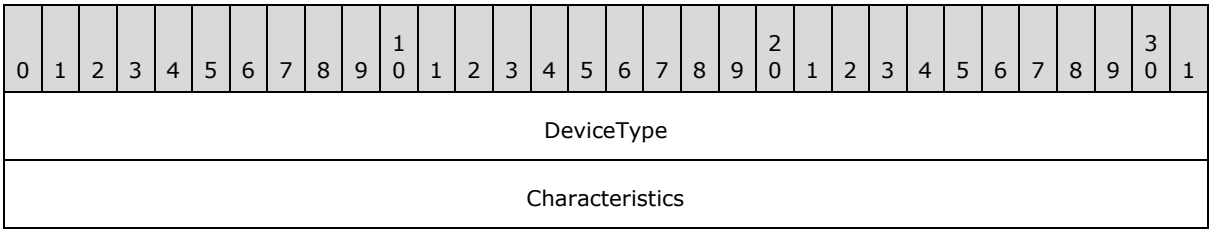
This operation returns a status code, as specified in [MS-ERREF] section 2.3. The status code returned directly by the function that processes this file information class MUST be STATUS_SUCCESS or one of the following.

Error code	Meaning
STATUS_INFO_LENGTH_MISMATCH 0xC0000004	The specified information record length does not match the length that is required for the specified information class.
STATUS_BUFFER_OVERFLOW 0x80000005	The output buffer was filled before all of the volume information could be returned; only a portion of the **VolumeLabel** field is returned.

2.5.9 FileFsDeviceInformation

This information class is used to query device information associated with a file system volume.

A FILE_FS_DEVICE_INFORMATION data element, defined as follows, is returned by the server.



DeviceType (4 bytes): This identifies the type of given volume. It MUST be one of the following.

Value	Meaning
FILE_DEVICE_CD_ROM 0x00000002	Volume resides on a CD ROM.
FILE_DEVICE_DISK 0x00000007	Volume resides on a disk.

Characteristics (4 bytes): A bit field which identifies various characteristics about a given volume. The following are valid bit values.

Value	Meaning
FILE_REMOVABLE_MEDIA 0x00000001	Indicates that the storage device supports removable media. Notice that this characteristic indicates removable media, not a removable device. For example, drivers for JAZ drive devices specify this characteristic, but drivers for PCMCIA flash disks do not.
FILE_READ_ONLY_DEVICE	Indicates that the device cannot be written to.

Value	Meaning
0x00000002	
FILE_FLOPPY_DISKETTE 0x00000004	Indicates that the device is a floppy disk device.
FILE_WRITE_ONCE_MEDIA 0x00000008	Indicates that the device supports write-once media.
FILE_REMOTE_DEVICE 0x00000010	Indicates that the volume is for a remote file system like SMB or CIFS.
FILE_DEVICE_IS_MOUNTED 0x00000020	Indicates that a file system is mounted on the device.
FILE_VIRTUAL_VOLUME 0x00000040	Indicates that the volume does not directly reside on storage media, but resides on some other type of media (memory for example).
FILE_DEVICE_SECURE_OPEN 0x00000100	By default, volumes do not check the ACL associated with the volume, but instead use the ACLs associated with individual files on the volume. When this flag is set the volume ACL is also checked.
FILE_CHARACTERISTIC_WEBDAV_DEVICE 0x00002000	Indicates that a Web-based Distributed Authoring and Versioning (WebDAV) file system is mounted on the device. See [MS-WDVME] for more information.

This operation returns a status code, as specified in [MS-ERREF] section 2.3. The status code returned directly by the function that processes this file information class MUST be STATUS_SUCCESS or one of the following.

Error code	Meaning
STATUS_INFO_LENGTH_MISMATCH 0xC0000004	The specified information record length does not match the length that is required for the specified information class.

2.6 File Attributes

The following attributes are defined for files and directories. They can be used in any combination unless noted in the description of the attribute's meaning. There is no file attribute with the value 0x00000000 because a value of 0x00000000 in the **FileAttributes** field means that the file attributes for this file MUST NOT be changed when setting basic information for the file.

Value	Meaning
FILE_ATTRIBUTE_ARCHIVE 0x00000020	A file or directory that requires to be archived. Applications use this attribute to mark files for backup or removal.
FILE_ATTRIBUTE_COMPRESSED 0x00000800	A file or directory that is compressed. For a file, all of the data in the file is compressed. For a directory, compression is the default for newly created files and subdirectories.
FILE_ATTRIBUTE_DIRECTORY	This item is a directory.

Value	Meaning
0x00000010	
FILE_ATTRIBUTE_ENCRYPTED 0x00004000	A file or directory that is encrypted. For a file, all data streams in the file are encrypted. For a directory, encryption is the default for newly created files and subdirectories.
FILE_ATTRIBUTE_HIDDEN 0x00000002	A file or directory that is hidden. Files and directories marked with this attribute do not appear in an ordinary directory listing.
FILE_ATTRIBUTE_NORMAL 0x00000080	A file that does not have other attributes set. This flag is used to clear all other flags by specifying it with no other flags set. This flag MUST be ignored if other flags are set.<130>
FILE_ATTRIBUTE_NOT_CONTENT_INDEXED 0x00002000	A file or directory that is not indexed by the content indexing service.
FILE_ATTRIBUTE_OFFLINE 0x00001000	The data in this file is not available immediately. This attribute indicates that the file data is physically moved to offline storage. This attribute is used by Remote Storage, which is hierarchical storage management software.
FILE_ATTRIBUTE_READONLY 0x00000001	A file or directory that is read-only. For a file, applications can read the file but cannot write to it or delete it. For a directory, applications cannot delete it, but applications can create and delete files from that directory.
FILE_ATTRIBUTE_REPARSE_POINT 0x00000400	A file or directory that has an associated reparse point.
FILE_ATTRIBUTE_SPARSE_FILE 0x00000200	A file that is a sparse file.
FILE_ATTRIBUTE_SYSTEM 0x00000004	A file or directory that the operating system uses a part of or uses exclusively.
FILE_ATTRIBUTE_TEMPORARY 0x00000100	A file that is being used for temporary storage. The operating system may choose to store this file's data in memory rather than on mass storage, writing the data to mass storage only if data remains in the file when the file is closed.

3 Structure Examples

For structure examples, see the individual protocols (such as the Distributed Link Tracking: Workstation Protocol; for more information, see [MS-DLTW] section 3.1.6) that use the structures and constants defined in this document.

4 Security Considerations

4.1 Security Considerations for Implementers

Allowing the use of native information levels and file system controls by a protocol could unintentionally grant access to a wider range of functionality than the protocol author intended. Developers who choose to take advantage of these common structures in a generic format should protect their applications appropriately by blocking both the levels that they do not want to support and the levels that they do not expect.

For example, the protocol could verify that the provided level is within the range of levels that existed at the time of protocol design and development before the protocol performs any further processing. The latter is significant if the underlying file system might be upgraded to support new functionality that was not there when the protocol was initially implemented.

4.2 Index of Security Parameters

None.

5 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 NT® operating system
- 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

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 2.1.2.1: All reparse tags defined by Microsoft components MUST have the high bit set to 1. Non-Microsoft reparse tags MUST have the high bit set to 0.

<2> Section 2.1.2.1: These are Microsoft reparse tags. Except where explicitly allowed, clients MUST NOT process the Microsoft reparse tag data buffers.

<3> Section 2.1.2.1: The Windows Home Server Drive Extender is part of the Windows Home Server product.

<4> Section 2.1.2.1: The filter manager test harness is not shipped with Windows.

<5> Section 2.1.3.1: When a file is moved or copied from one volume to another, the **ObjectId** member value changes to a random unique value to avoid the potential for **ObjectId** collisions because the object ID is not guaranteed to be unique across volumes.

<6> Section 2.1.3.1: The NTFS file system places no constraints on the format of the 48 bytes of information following the ObjectId in this structure. This format of the FILE_OBJECTID_BUFFER is used on Windows by the Microsoft Distributed Link Tracking Service (see [MS-DLTW] section 3.1.6).

<7> Section 2.1.3.2: Windows places Distributed Link Tracking information into the ExtendedInfo field for use by the Distributed Link Tracking protocols (see [MS-DLTW] section 3.1.6).

<8> Section 2.1.4: Two file systems supported by Windows, NTFS and **Universal Disk Format (UDF)**, provide alternate data stream functionality.

<9> Section 2.1.8: Windows defines a TRUE as "1"; however, it will interpret any nonzero value as TRUE.

<10> Section 2.2: Reparse points, object IDs, and the update sequence number (USN) change journal are supported by NTFS, but not by the Microsoft FAT file system. Therefore, FSCTLs involving these technologies will return STATUS_INVALID_DEVICE_REQUEST when the specified file or directory is located on a volume formatted with the FAT file system. Windows also returns STATUS_INVALID_DEVICE_REQUEST when a required file system filter is supported by the file system but is not installed (see section 2.3.62).

<11> Section 2.2: The following table lists FSCTLs that are not supported remotely and that, if received by the object store, will respond with a status code other than STATUS_INVALID_DEVICE_REQUEST, as specified in section 2.2.

FSCTL name	FSCTL function number	Status Code
FSCTL_GET_BOOT_AREA_INFO	0x90230	STATUS_INVALID_PARAMETER
FSCTL_GET_RETRIEVAL_POINTER_BASE	0x90234	STATUS_INVALID_PARAMETER
FSCTL_IS_VOLUME_DIRTY	0x90078	STATUS_INVALID_PARAMETER
FSCTL_ALLOW_EXTENDED_DASD_IO	0x90083	STATUS_ACCESS_DENIED
FSCTL_LOOKUP_STREAM_FROM_CLUSTER	0x901FC	STATUS_INVALID_PARAMETER
FSCTL_EXTEND_VOLUME	0x900F0	STATUS_INVALID_PARAMETER
FSCTL_SHRINK_VOLUME	0x901B0	STATUS_INVALID_PARAMETER
FSCTL_FILE_PREFETCH	0x90120	STATUS_INVALID_PARAMETER
FSCTL_SET_SHORT_NAME_BEHAVIOR	0x901B4	STATUS_ACCESS_DENIED
FSCTL_SET_PERSISTENT_VOLUME_STATE	0x90238	STATUS_INVALID_PARAMETER
FSCTL_QUERY_PERSISTENT_VOLUME_STATE	0x9023C	STATUS_INVALID_PARAMETER
FSCTL_SD_GLOBAL_CHANGE	0x901F4	STATUS_INVALID_PARAMETER

<12> Section 2.3: The NtFsControlFile function is used to invoke an FSCTL on a file handle. The definition of this function, including its content and the function signature, is implementation-dependent, and is not part of the protocol specification.

<13> Section 2.3.2: Windows will try 16 times to generate a unique ID, and will fail with this status if 16 attempts have been unsuccessful.

<14> Section 2.3.7: This FSCTL is implemented on **NTFS**, **FAT**, and **exFAT** file systems. Other file systems return STATUS_INVALID_DEVICE_REQUEST.

<15> Section 2.3.8: This FSCTL is implemented on **NTFS**, **FAT**, and **exFAT** file systems. Other file systems return STATUS_INVALID_DEVICE_REQUEST.

<16> Section 2.3.10: NTFS always returns at least 2 bytes and up to 8 bytes of trailing padding after each entry in the reply, including the last entry.

<17> Section 2.3.12: The LZNT1 is the only compression algorithm implemented on Windows 2000, Windows XP, Windows Server 2003, Windows Vista, Windows Server 2008, Windows 7, and Windows Server 2008 R2.

<18> Section 2.3.12: Windows 2000, Windows XP, Windows Server 2003, Windows Vista, Windows Server 2008, Windows 7, and Windows Server 2008 R2 support file compression on volumes that are formatted with the NTFS file system and have a cluster size less than or equal to 4 kilobytes.

<19> Section 2.3.18:

- Windows NT 4.0 returns STATUS_INVALID_DEVICE_REQUEST for a file on an NTFS, FAT, or CDFS file system.
- Windows 2000 returns STATUS_INVALID_DEVICE_REQUEST for a file on a FAT or CDFS file system.
- Windows XP returns STATUS_INVALID_DEVICE_REQUEST for a file on a FAT or CDFS file system.
- Windows Server 2003 returns STATUS_INVALID_DEVICE_REQUEST for a file on a FAT or CDFS file system.
- Windows Vista returns STATUS_INVALID_DEVICE_REQUEST for a file on a FAT or CDFS file system.
- Windows Server 2008 returns STATUS_INVALID_DEVICE_REQUEST for a file on a FAT or CDFS file system.
- Windows 7 returns STATUS_INVALID_DEVICE_REQUEST for a file on a FAT or CDFS file system.
- Windows Server 2008 R2 returns STATUS_INVALID_DEVICE_REQUEST for a file on a FAT or CDFS file system.

<20> Section 2.3.20: On an NTFS volume, very short data streams (typically several hundred bytes) can be written to disk without having any clusters allocated. These short streams are sometimes called resident because the data resides in the file's master file table (MFT) record. A resident data stream has no retrieval pointers to return.

<21> Section 2.3.22: Windows NT, Windows 2000, Windows XP, Windows Server 2003, Windows Vista, Windows Server 2008, Windows 7, and Windows Server 2008 R2 support the FSCTL_IS_PATHNAME_VALID Request (section 2.3.21) and return STATUS_SUCCESS whenever this request is invoked.

<22> Section 2.3.32: Each entry in the output array contains an offset and a length that indicates a range in the file that may contain nonzero data. The actual nonzero data, if any, is somewhere within this range, and the calling application must scan further within the range to locate it and determine if it really is valid data. Multiple instances of valid data may exist within the range.

<23> Section 2.3.32: Sparse files are supported by Windows 2000, Windows XP, Windows Server 2003, Windows Vista, Windows Server 2008, Windows 7, and Windows Server 2008 R2. The NTFS file system rounds down the input file offset to a 65,536-byte (64-kilobyte) boundary, rounds up the length to a convenient boundary, and then begins to walk through the file.

<24> Section 2.3.32: Windows does not track every piece of zero (0) or nonzero data. Because zero (0) is often perfectly legal data, it would be misleading. Instead, the system tracks ranges in which disk space is allocated. Where no disk space is allocated, all data bytes within that range for **Length** bytes from **FileOffset** are assumed to be zero (0) (when data is read, NTFS returns a zero for every byte in a sparse region). Allocated storage can contain zero (0) or nonzero data. So all that this operation does is return information on parts of the file where nonzero data might be located. It is up to the application to scan these parts of the file in accordance with the application's data conventions.

<25> Section 2.3.36: The following is the Windows UDF File System Support table. It lists the UDF revisions and "builds" (VAT/Spared/Write) that are supported by each covered version of Windows.

Windows	UDF V1.02	UDF V1.5	UDF V2.01	UDF V2.5	UDF 2.6
95 / 95OSR2	-	-	-	-	-
98	Read	-	-	-	-
NT4	-	-	-	-	-
2000	Read	Read	-	-	-
XP	Read	Read	Read	-	-
W2K3	Read	Read	Read	-	-
Vista	Read/Write	Read/Write	Read/Write	Read/Write	-
Windows 7	Read/Write	Read/Write	Read/Write	Read/Write	Read/Write

Note If Read of a given UDF version is supported, then reading of all UDF variants of that version are supported (VAT, Sparing and Simple). If Read/Write of a given UDF version is supported, then reading/writing of all UDF variants of that version are supported (VAT, Sparing and Simple).

<26> Section 2.3.36: The Windows UDF implementation pads the entire **CopyrightInfo** field with NULLs.

<27> Section 2.3.36: The Windows UDF implementation pads the entire **AbstractInfo** field with NULLs.

<28> Section 2.3.36: When the volume is formatted on Windows, this value is set to "*Microsoft Windows" followed by Unicode NULLs.

<29> Section 2.3.36: When the volume is written to on a Windows system, this value is set to "*Microsoft Windows" followed by Unicode NULLs.

<30> Section 2.3.40: The major version number is 2 for file systems created on Windows 2000, Windows XP, Windows Server 2003, Windows Vista, Windows Server 2008, Windows 7, and Windows Server 2008 R2.

<31> Section 2.3.40: The minor version number is 0 for file systems created on Windows 2000, Windows XP, Windows Server 2003, Windows Vista, Windows Server 2008, Windows 7, and Windows Server 2008 R2.

<32> Section 2.3.40: Windows computes the file reference number as follows: 48 bits are the index of the file's primary record in the master file table (MFT), and the other 16 bits are a sequence number. Therefore, it is possible that a different file can have the same FileReferenceNumber as a file on that volume had in the past; however, this is an unlikely scenario.

<33> Section 2.3.40: The contents of a USN_RECORD element returned by this FSCTL is a partially populated record compared to the fully populated records returned by local only FSCTL FSCTL_READ_USN_JOURNAL.

<34> Section 2.3.43: Equivalent to COMPRESSION_FORMAT_LZNT1.

<35> Section 2.3.43: The LZNT1 is the only compression algorithm implemented on Windows 2000, Windows XP, Windows Server 2003, Windows Vista, Windows Server 2008, Windows 7, and Windows Server 2008 R2. Therefore, requests for COMPRESSION_FORMAT_DEFAULT and COMPRESSION_FORMAT_LZNT1 are equivalent from the server's perspective.

<36> Section 2.3.47: This message is implemented only on NTFS, and it is only for private use by the Encrypted File System (EFS). EFS issues this message locally on the machine that physically contains the file, notifying NTFS of a change in the file/stream attributes and causing NTFS to invoke the EFS callback that does the actual work of encrypting/decrypting streams.

This message is not used by any other component other than local EFS on Windows. It is not sent by the SMB1 and SMB2 client redirectors, nor is it accepted by an SMB2 server. In order to manipulate the encryption state of files and streams, clients use EFS and the EFSRPC protocol specified in [MS-EFSR].

<37> Section 2.3.47: The SMB1 server does not currently fail the FSCTL_SET_ENCRYPTION Request (section 2.3.47) if received. A QFE is planned to address this issue for the SMB1 server.

<38> Section 2.3.47: Windows sets the FILE_ATTRIBUTE_ENCRYPTED flag in the duplicate information file attributes field, and invokes the EFS callback which then creates the $EFS attribute.

<39> Section 2.3.47: Windows takes the following actions to clear encryption:

- Clears the FILE_ATTRIBUTE_ENCRYPTED flag in the duplicate information file attributes field.
- Invokes the EFS callback, which removes the $EFS attribute.

<40> Section 2.3.47: Windows takes the following actions to set encryption on a stream:

- If the stream is a resident user data stream, converts it to non-resident.
- Sets ATTRIBUTE_FLAG_ENCRYPTED in the attribute header.
- Invokes the EFS callback to generate an encryption context for this stream.

Note that if this is called during the creation of a named data attribute on a file with an empty unnamed data attribute, then the unnamed data attribute will be converted to non-resident and its attribute header flag will be set to encrypted.

Also note that this will set the FILE_ATTRIBUTE_ENCRYPTED flag if it is the first stream on the file that is encrypted.

<41> Section 2.3.47: Windows clears the ATTRIBUTE_FLAG_ENCRYPTED flag from the attribute header and invokes the EFS callback to free the encryption context for the stream.

<42> Section 2.3.47: The **Private** field is a placeholder marking the beginning of the private portion of the encryption buffer structure. This portion of the structure is meaningful only to EFS, because all the information necessary to fill (making a well-formed request) is private to EFS. Windows uses the EFSRPC protocol as specified in [MS-EFSR] to manipulate file encryption state.

<43> Section 2.3.48: An FSCTL_SET_ENCRYPTION operation should never succeed unless it is requested by the Encrypted File System (EFS), because the information necessary to make a well-formed request is visible only to EFS, as FSCTL_SET_ENCRYPTION is only for private use by EFS. Windows uses the EFSRPC protocol as specified in [MS-EFSR] to manipulate file encryption state.

<44> Section 2.3.48: On Windows, encryption requires NTFS major version 2 or greater.

<45> Section 2.3.48: Windows returns this error code if the NTFS encryption driver is not loaded or the FILE_CLEAR_ENCRYPTION operation was requested on a file containing a stream that is still marked as encrypted.

<46> Section 2.3.48: Windows returns this error code if the $INDEX_ROOT attribute of the directory that was trying to be encrypted, could not be found.

<47> Section 2.3.49: Windows expects that the file whose object identifier is set with this FSCTL has been opened for write and that backup/restore operations were specified at file open. In Windows, this is accomplished by specifying the flag, FILE_FLAG_BACKUP_SEMANTICS (whose value is 0x02000000), along with other attributes such as FILE_ATTRIBUTE_NORMAL when opening the file.

<48> Section 2.3.49: All Windows versions: This request is never sent to a remote server.

<49> Section 2.3.51: The Microsoft Distributed Link Tracking Service uses the last 48 bytes of the ExtendedInfo BLOB to store information that helps it locate files that are moved to different volumes or computers within a domain. For more information, see [MS-DLTW] section 3.1.6.

<50> Section 2.3.55: NTFS does not attempt to recover a failed unsparse operation by "resparsing".

<51> Section 2.3.55: NFTS does not deallocate existing clusters.

<52> Section 2.3.57: On receipt of this message, NTFS might deallocate disk space in the file if the file is stored on an NTFS volume, and the file is sparse or compressed. It will free any allocated space in chunks of 64 kilobytes that begin at an offset that is a multiple of 64 kilobytes. Other bytes in the file (prior to the first freed 64-kilobyte chunk and after the last freed 64-kilobyte chunk) will be zeroed but not deallocated. This FSCTL sets the range of bytes to zeros (0) without extending the file size.

<53> Section 2.3.59: This message is implemented only by NTFS, which is supported on Windows NT, Windows XP, Windows 2000, Windows Server 2003, Windows Vista, Windows Server 2008, Windows 7, and Windows Server 2008 R2.

<54> Section 2.3.61: Both the source and destination file names must represent paths on the same volume, and the file names are the full paths to the files, including the share or drive letter at which each file is located.

<55> Section 2.4: Windows uses the NtQueryInformationFile function to process the specified query for file information and NtSetInformationFile to process the specified request to set file information. The definition of the function used to process any file information request, including its content and the function signature, is implementation-dependent and is not part of the protocol specification.

<56> Section 2.4: This file information class is intended for local use only; the server will fail it with STATUS_NOT_SUPPORTED.

<57> Section 2.4: This file information class is intended for local use only; the server will fail it with STATUS_NOT_SUPPORTED.

<58> Section 2.4: This file information class is intended for local use only; the server will fail it with STATUS_NOT_SUPPORTED.

<59> Section 2.4: This file information class is intended for local use only; the server will fail it with STATUS_NOT_SUPPORTED.

<60> Section 2.4: This information class is not sent across the wire. In Windows, it is handled by the **IOManager** on the client. If this operation is sent to an SMB server, both SMB and SMB2 send the request to the **IOManager** on the server and perform normal processing of the operation.

<61> Section 2.4: Windows file systems do not implement this file information class; the server will fail it with STATUS_NOT_SUPPORTED.

<62> Section 2.4: This file information class is intended for local use only; the server will fail it with STATUS_NOT_SUPPORTED.

<63> Section 2.4: Windows file systems do not implement this file information class; the server will fail it with STATUS_NOT_SUPPORTED.

<64> Section 2.4: This file information class is intended for local use only; the server will fail it with STATUS_NOT_SUPPORTED.

<65> Section 2.4: The CIFS, SMB, and SMB2 protocols do not directly call this information class but use the structures associated with it.

<66> Section 2.4: This file information class is intended for local use only; the server will fail it with STATUS_NOT_SUPPORTED.

<67> Section 2.4: This file information class is intended for local use only; the server will fail it with STATUS_NOT_SUPPORTED.

<68> Section 2.4: Windows file systems do not implement this file information class; the server will fail it with STATUS_NOT_SUPPORTED.

<69> Section 2.4: This file information class is intended for local use only; the server will fail it with STATUS_NOT_SUPPORTED.

<70> Section 2.4: This file information class is intended for local use only; the server will fail it with STATUS_NOT_SUPPORTED.

<71> Section 2.4.4: A file's allocation size and end-of-file position are independent of each other with the following exception: The end-of-file position must always be less than or equal to the allocation size. If the allocation size is set to a value that is less than the end-of-file position, the end-of-file position is automatically adjusted to match the allocation size. Because the end-of-file position may be less than the file's allocation size, the last sector (or cluster) of a file can have unused bytes between the last byte of the file and the last byte of the sector (or cluster).

<72> Section 2.4.4: NTFS rounds allocation size for resident files to a multiple of 8 bytes. When shrinking a resident file’s allocation size using the FileAllocationInformation info class, the file remains resident with an allocation size rounded up to a multiple of 8 bytes. When extending a resident file’s allocation size using the FileAllocationInformation info class, the file is converted to nonresident with an allocation size rounded up to a multiple of the cluster size.

<73> Section 2.4.5: NTFS assigns an alternate name to a file whose full name is not compliant with restrictions for file names under MS-DOS and 16-bit Windows unless the system has been configured through a registry entry to not generate these names to improve performance.

<74> Section 2.4.7: The file system updates the values of the **LastAccessTime**, **LastWriteTime**, and **ChangeTime** members as appropriate after an I/O operation is performed on a file. However, a driver or application can request that the file system not update one or more of these members for I/O operations that are performed on the caller's file handle by setting the appropriate members to -1. The caller can set one, all, or any other combination of these three members to -1. Only the members that are set to -1 will be unaffected by I/O operations on the file handle; the other

members will be updated as appropriate. This behavior is consistent across all file system types. Note that even though -1 can be used with the **CreationTime** field, it has no effect because file creation time is never updated in response to file system calls such as read and write.

<75> Section 2.4.7: The file system updates the values of the **LastAccessTime**, **LastWriteTime**, and **ChangeTime** members as appropriate after an I/O operation is performed on a file. However, a driver or application can request that the file system not update one or more of these members for I/O operations that are performed on the caller's file handle by setting the appropriate members to -1. The caller can set one, all, or any other combination of these three members to -1. Only the members that are set to -1 will be unaffected by I/O operations on the file handle; the other members will be updated as appropriate. This behavior is consistent across all file system types. Note that even though -1 can be used with the **CreationTime** field, it has no effect because file creation time is never updated in response to file system calls such as read and write.

<76> Section 2.4.7: The file system updates the values of the **LastAccessTime**, **LastWriteTime**, and **ChangeTime** members as appropriate after an I/O operation is performed on a file. However, a driver or application can request that the file system not update one or more of these members for I/O operations that are performed on the caller's file handle by setting the appropriate members to -1. The caller can set one, all, or any other combination of these three members to -1. Only the members that are set to -1 will be unaffected by I/O operations on the file handle; the other members will be updated as appropriate. This behavior is consistent across all file system types. Note that even though -1 can be used with the **CreationTime** field, it has no effect because file creation time is never updated in response to file system calls such as read and write.

<77> Section 2.4.8: When using NTFS, the position of a file within the parent directory is not fixed and can be changed at any time. Windows 2000, Windows XP, Windows Server 2003, Windows Vista, Windows Server 2008, Windows 7, and Windows Server 2008 R2 set this value to zero for files on NTFS file systems.

<78> Section 2.4.9: Windows 2000, Windows XP, Windows Server 2003, Windows Vista, Windows Server 2008, Windows 7, and Windows Server 2008 R2 implement only one compression algorithm, LZNT1. For more information, see [UASDC].

<79> Section 2.4.9: NTFS uses a value of 16 calculated as (4 + **ClusterShift**) for the CompressionUnitShift by default. The ultimate size of data to be compressed depends on the cluster size set for the file system at initialization. NTFS defaults to a 4-kilobyte cluster size, resulting in a **ClusterShift** value of 12, but NTFS file systems can be initialized with a different cluster size, so the value may vary. The default compression unit size based on this calculation is 64 kilobytes.

<80> Section 2.4.9: NTFS uses a value of 12 for the ChunkShift so that compression chunks are 4 kilobytes in size.

<81> Section 2.4.9: The value of this field depends on the cluster size set for the file system at initialization. NTFS uses a value of 12 by default because the default NTFS cluster size is 4 kilobytes. If an NTFS file system is initialized with a different cluster size, the value of **ClusterShift** would be log 2 of the cluster size for that file system.

<82> Section 2.4.10: When using NTFS, the position of a file within the parent directory is not fixed and can be changed at any time. Windows 2000, Windows XP, Windows Server 2003, Windows Vista, Windows Server 2008, Windows 7, and Windows Server 2008 R2 set this value to zero for files on NTFS file systems.

<83> Section 2.4.11: A file marked for deletion is not actually deleted until all open handles for the file object have been closed, and the link count for the file is zero.

<84> Section 2.4.14: When using NTFS, the position of a file within the parent directory is not fixed and can be changed at any time. Windows 2000, Windows XP, Windows Server 2003, Windows Vista, Windows Server 2008, Windows 7, and Windows Server 2008 R2 set this value to zero for files on NTFS file systems.

<85> Section 2.4.16: In Windows, both the NTFS and UDFS file systems support hard links. UDFS support of hard links was added in Windows Vista and Windows Server 2008.

<86> Section 2.4.17: When using NTFS, the position of a file within the parent directory is not fixed and can be changed at any time. Windows 2000, Windows XP, Windows Server 2003, Windows Vista, Windows Server 2008, Windows 7, and Windows Server 2008 R2 set this value to zero for files on NTFS file systems.

<87> Section 2.4.18: Windows-based SMB Version 1 servers set the **NextEntryOffset** field to the size of the current **FileIdFullDirectoryInformation** entry in bytes, if no other entries follow this one.

<88> Section 2.4.18: When using NTFS, the position of a file within the parent directory is not fixed and can be changed at any time. Windows 2000, Windows XP, Windows Server 2003, Windows Vista, Windows Server 2008, Windows 7, and Windows Server 2008 R2 set this value to zero for files on NTFS file systems.

<89> Section 2.4.21: In Windows, both the NTFS and UDFS file systems support hard links. UDFS support of hard links was added in Windows Vista and Windows Server 2008.

<90> Section 2.4.24: This information class is not sent across the wire. In Windows, it is handled by the **IOManager** on the client. If this operation is sent to an SMB server, both SMB and SMB2 send the request to the **IOManager** on the server and perform normal processing of the operation.

<91> Section 2.4.24: This flag is cleared by the respective server application while processing the set operation in the following situations:

- SMB server on all supported versions of Windows if the file is not opened with a **DesiredAccess** field value that has the FILE_WRITE_DATA or FILE_APPEND_DATA bit set (see [MS-CIFS] section 2.2.4.64.1).
- SMB2 server on Windows Vista and Windows Server 2008 always.
- SMB2 server on Windows 7 and Windows Server 2008 R2 if the file is opened with a **CreateOptions** field value that has the FILE_NO_INTERMEDIATE_BUFFERING bit set (see [MS-SMB2] section 2.2.13).

<92> Section 2.4.26: When using NTFS, the position of a file within the parent directory is not fixed and can be changed at any time. Windows 2000, Windows XP, Windows Server 2003, Windows Vista, Windows Server 2008, Windows 7, and Windows Server 2008 R2 set this value to zero for files on NTFS file systems.

<93> Section 2.4.27: This operation works on both remote and local handles.

<94> Section 2.4.28: The Microsoft FAT file system does not support the use of ObjectIds, and returns a status code of STATUS_INVALID_DEVICE_REQUEST.

<95> Section 2.4.28: The Microsoft Distributed Link Tracking protocols (see [MS-DLTW] section 3.1.6) use the first type of object ID structure for link tracking.

<96> Section 2.4.28.1: When a file is moved or copied from one volume to another, the **ObjectId** member's value changes to a random unique value to avoid the potential for **ObjectId** collisions because the object ID is not guaranteed to be unique across volumes.

<97> Section 2.4.32: Both the query and set FilePositionInformation operations are processed on the local client; therefore, these operations are not transmitted across the wire. The fact that these operations are processed on the client instead of the server is intended to be transparent to the client's usage of these operations.

If a server receives a request to set FilePositionInformation, the specified file position will be set on the remote handle, but its value will be ignored by future read/write operations. If a server receives a request to query FilePositionInformation, an undetermined value will be returned. For more information on how the **CurrentByteOffset** field is updated, see the [MS-FSA] sections for read and write operations.

<98> Section 2.4.32: Each read and write operation via the Server Message Block (SMB) Protocol [MS-SMB] and Server Message Block (SMB) Version 2 [MS-SMB2] protocols always provides an explicit starting offset, and thus is unaffected by the file position. Windows does not update the file position when read and write operations are performed via these protocols.

<99> Section 2.4.33: Query and set operations are supported only by the NTFS file system and are valid only on handles opened to the NTFS metadata file "\$Extend\$Quota:$Q:$INDEX_ALLOCATION".

<100> Section 2.4.37: In Windows 7 and Windows Server 2008 R2, the existing short name is deleted if the **FileNameLength** field in **FILE_NAME_INFORMATION** is zero. Previous Windows implementations return STATUS_INVALID_PARAMETER when the **FileNameLength** field is zero.

<101> Section 2.4.39: This information class is supported on Windows 7 and Windows Server 2008 R2.

<102> Section 2.4.41: Windows supports the FileValidDataLengthInformation (section 2.4.41) information class in the NTFS, FAT, and EXFAT file systems.

<103> Section 2.5: Windows uses the NtQueryVolumeInformationFile function to process the specified query for file system information and the NtSetVolumeInformationFile function to set the specified file system information. The definition of the function used to process any file system information request, including its content and the function signature, is implementation-dependent and is not part of the protocol specification.

<104> Section 2.5: This file system information class is intended for local use only; the server will fail it with status STATUS_NOT_SUPPORTED.

<105> Section 2.5: This file system information class is intended for local use only; the server will fail it with status STATUS_NOT_SUPPORTED. Furthermore, this file information class is not implemented by any Windows file systems.

<106> Section 2.5: This file system information class is intended for local use only; the server will fail a "query" with STATUS_ACCESS_NOT_SUPPORTED, and the server will fail a "set" with STATUS_ACCESS_DENIED. Furthermore, this file information class is not implemented by any Windows file systems.

<107> Section 2.5.1: This attribute is only available on Windows 7 and Windows Server 2008 R2.

<108> Section 2.5.1: This attribute is only available on Windows 7 and Windows Server 2008 R2.

<109> Section 2.5.1: This attribute is only available on Windows 7 and Windows Server 2008 R2.

<110> Section 2.5.1: This attribute is only available on Windows 7 and Windows Server 2008 R2.

<111> Section 2.5.1: Windows Vista, Windows Server 2008, Windows 7, and Windows Server 2008 R2 set this flag if the volume is formatted for NTFS 3.0 or higher.

<112> Section 2.5.1: This attribute is only available on Windows XP, Windows Server 2003, Windows Vista, Windows Server 2008, Windows 7, and Windows Server 2008 R2.

<113> Section 2.5.1: Windows support for a volume formatted to NTFS version 3.0 or 3.1 is required for EFS use. NTFS versions 3.0 and 3.1 are supported on Windows 2000, Windows XP, Windows Server 2003, Windows Vista, Windows Server 2008, Windows 7, and Windows Server 2008 R2.

<114> Section 2.5.1: Remote storage is provided by the Remote Storage service to create virtual disk storage from a tape or other storage media.

<115> Section 2.5.1: For the Microsoft NTFS, FAT, and EXFAT file systems, this value is 510. For the Microsoft UDFS file system, this value is 508. For the Microsoft CDFS file system, this value is 220 for Joliet format and 442 otherwise.

<116> Section 2.5.1: Valid values for this field depend on the version of Windows that the server is running. For Windows 7, valid values are "FAT", "FAT16", "FAT32", "exFAT", "NTFS", "CDFS", "UDF". For Windows Vista SP1, Windows Server 2008, and Windows Server 2008 R2, valid values are "FAT", "FAT16", "FAT32", "exFAT", "NTFS", "CDFS", and "UDF". For Windows Vista RTM, valid values are: "FAT", "FAT16", "FAT32", "NTFS", "CDFS", and "UDF". For Windows XP, valid values are "FAT", "FAT16", "FAT32", "NTFS", and "CDFS".

<117> Section 2.5.2: Windows sets this value to zero.

<118> Section 2.5.2: Windows sets this value to zero.

<119> Section 2.5.2: Windows sets this value to zero.

<120> Section 2.5.2: Logging makes an entry in the Windows application event log.

<121> Section 2.5.4: In Windows 2000, Windows XP, Windows Server 2003, Windows Vista, Windows Server 2008, Windows 7, and Windows Server 2008 R2, if per-user quotas are in use, this value may be less than the total number of allocation units on the disk. Non-Microsoft quota management software might display the same behavior as these versions of Windows if that software was implemented as a file system filter driver, and the driver implementer opted to set the FileFsFullSizeInformation in the same manner as Windows 2000.

<122> Section 2.5.4: In Windows 2000, Windows XP, Windows Server 2003, Windows Vista, Windows Server 2008, Windows 7, and Windows Server 2008 R2, if per-user quotas are in use, this value may be less than the total number of free allocation units on the disk.

<123> Section 2.5.5: A maximum length of 32 characters is imposed for any Windows file system, though some file systems may impose a stricter limit. The Microsoft FAT file system supports volume labels that are 0 to 11 characters in length. NTFS supports volume labels that are 0 to 32 characters in length. All Unicode characters are permitted in a volume label with the exception of the NULL character, which is reserved for use as a string terminator.

<124> Section 2.5.6: The Microsoft FAT, EXFAT, UDFS, or CDFS file systems do not support the use of object IDs, and return a status code of STATUS_INVALID_DEVICE_REQUEST.

<125> Section 2.5.6: Windows does not write information into the **ExtendedInfo** field for file systems.

<126> Section 2.5.7: In Windows 2000, Windows XP, Windows Server 2003, Windows Vista, Windows Server 2008, Windows 7, and Windows Server 2008 R2, if per-user quotas are in use, this value may be less than the total number of allocation units on the disk. Non-Microsoft quota management software might display the same behavior as Windows 2000 if that software was implemented as a file system filter driver, and the driver implementer opted to set the FileFsSizeInformation in the same manner as Windows 2000.

<127> Section 2.5.7: In Windows 2000, Windows XP, Windows Server 2003, Windows Vista, Windows Server 2008, Windows 7, and Windows Server 2008 R2, if per-user quotas are in use, this value may be less than the total number of free allocation units on the disk.

<128> Section 2.5.8: A maximum length of 32 characters is imposed for any Windows file system, though some file systems may impose a stricter limit. The Microsoft FAT file system supports volume labels that are 0 to 11 characters in length. NTFS supports volume labels that are 0 to 32 characters in length. All Unicode characters are permitted in a volume label with the exception of the NULL character, which is reserved for use as a string terminator.

<129> Section 2.5.8: This value is TRUE for NTFS and FALSE for other file systems implemented by Windows.

<130> Section 2.6: The Windows file system does not persist the FILE_ATTRIBUTE_NORMAL flag. When getting attributes via the FileAttributeTagInformation (section 2.4.6) information class, a client will receive the FILE_ATTRIBUTE_NORMAL flag only if no other attributes were set. Some examples: If a client sets the attributes as [FILE_ATTRIBUTE_HIDDEN | FILE_ATTRIBUTE_NORMAL], the client will see just [FILE_ATTRIBUTE_HIDDEN] when it gets the attributes. If the client sets the attributes as [FILE_ATTRIBUTE_NORMAL], the client will see [FILE_ATTRIBUTE_NORMAL] when it gets the attributes.

6 Change Tracking

This section identifies changes that were made to the [MS-FSCC] protocol document between the May 2011 and June 2011 releases. Changes are classified as New, Major, Minor, Editorial, or No change.

The revision class **New** means that a new document is being released.

The revision class **Major** means that the technical content in the document was significantly revised. Major changes affect protocol interoperability or implementation. Examples of major changes are:

- A document revision that incorporates changes to interoperability requirements or functionality.
- An extensive rewrite, addition, or deletion of major portions of content.
- The removal of a document from the documentation set.
- Changes made for template compliance.

The revision class **Minor** means that the meaning of the technical content was clarified. Minor changes do not affect protocol interoperability or implementation. Examples of minor changes are updates to clarify ambiguity at the sentence, paragraph, or table level.

The revision class **Editorial** means that the language and formatting in the technical content was changed. Editorial changes apply to grammatical, formatting, and style issues.

The revision class **No change** means that no new technical or language changes were introduced. The technical content of the document is identical to the last released version, but minor editorial and formatting changes, as well as updates to the header and footer information, and to the revision summary, may have been made.

Major and minor changes can be described further using the following change types:

- New content added.
- Content updated.
- Content removed.
- New product behavior note added.
- Product behavior note updated.
- Product behavior note removed.
- New protocol syntax added.
- Protocol syntax updated.
- Protocol syntax removed.
- New content added due to protocol revision.
- Content updated due to protocol revision.
- Content removed due to protocol revision.
- New protocol syntax added due to protocol revision.

- Protocol syntax updated due to protocol revision.
- Protocol syntax removed due to protocol revision.
- New content added for template compliance.
- Content updated for template compliance.
- Content removed for template compliance.
- Obsolete document removed.

Editorial changes are always classified with the change type **Editorially updated.**

Some important terms used in the change type descriptions are defined as follows:

- **Protocol syntax** refers to data elements (such as packets, structures, enumerations, and methods) as well as interfaces.
- **Protocol revision** refers to changes made to a protocol that affect the bits that are sent over the wire.

The changes made to this document are listed in the following table. For more information, please contact protocol@microsoft.com.

Section	Tracking number (if applicable) and description	Major change (Y or N)	Change type
1.2 References	Added explanatory statement regarding the removal of the publishing year from Microsoft Open Specification document references.	N	Content updated.

7 Index

G

I

L

M

N

O

P

R

S

T

V