[MS-FSCC]:
File System Control Codes

Revision Summary

Date	Revision History	Revision Class	Comments
04/03/2007	0.01		MCPP Milestone Longhorn Initial Availability
07/03/2007	1.0	Major	MLonghorn+90
07/20/2007	2.0	Major	Updated and revised the technical content.
08/10/2007	3.0	Major	Updated and revised the technical content.
09/28/2007	4.0	Major	Updated and revised the technical content.

Date	Revision History	Revision Class	Comments
10/23/2007	5.0	Major	Updated and revised the technical content.
11/30/2007	5.0.1	Editorial	Revised and edited the technical content.
01/25/2008	5.0.2	Editorial	Revised and edited the technical content.

Table of Contents

1 Introduction

This specification defines the network format of nativeWindows 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
FAT File System
Fid
FSCTL
Globally Unique Identifier (GUID)
NetBIOS Name
NTFS
Partition
Relative Identifier (RID)
Replica Set
Sector
Security Identifier (SID)
Stream
Update Sequence Number (USN)

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, 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.

Copy-on-Close: A method of handling writes to a file in which writes are not performed until the file has been closed. **Single-Instance Storage (SIS)** uses copy-on-close.

Direct Access to Storage Device (DASD) Handle: A handle to a disk device.

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**.

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

File Record Segment (FRS): 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 NT 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 provided by Microsoft as part of the Windows operating system that simplifies the development of non-Microsoft **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 Windows file system **filter manager**, which is an operating system component. A **filter** driver developed to the Windows file system **filter manager** model is called a **minifilter**.

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

Logical Cluster Number (LCN): The **cluster** number relative to the beginning of the **volume**. The first **cluster** on a **volume** is LCN 0.

LONGLONG: Signed 64-bit integer.

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.

Minifilter: A file system **filter** developed to work with the Windows 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, not by Microsoft. 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 resource 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. Reparse point tags are assigned by Microsoft. 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].

Single-Instance Storage (SIS): An **NTFS** feature that implements links with the semantics of copies for files stored on an **NTFS** **volume**. SIS uses **copy-on-close** to implement the copy semantics of its links. When using **copy-on-close**, the copy is delayed beyond the time of the first write until the complete set of updates is made to the file; and then, only the portions of the file that have not been written are copied from the SIS common store. For more information, see [SIS].

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.

Tag: Another name for a **reparse point**. For instance, the Windows 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.

Volume: A group of one or more **partitions** that forms a logical region of storage, and the basis for a file system. A volume is an area on a storage device that is managed by the file system as a discrete logical storage unit. A **partition** contains at least one volume, and a volume can exist on one or more **partitions**. A distinguishing feature of volumes is that they always have a root directory.

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

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", January 2007.

[MS-ERREF] Microsoft Corporation, "Windows Error Codes", January 2007.

[MS-GLOS] Microsoft Corporation, "Windows Protocols Master Glossary", March 2007.

[MS-LSAD] Microsoft Corporation, "Local Security Authority (Domain Policy) Remote Protocol Specification", June 2007.

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

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

[RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, March 1997, http://www.ietf.org/rfc/rfc2119.txt

1.2.2 Informative References

[CIFS] Leach, P. and Naik, D., "A Common Internet File System (CIFS/1.0) Protocol", March 1997, http://www.microsoft.com/about/legal/intellectualproperty/protocols/BSTD/CIFS/dr aft-leach-cifs-v1-spec-02.txt

If you have any trouble finding [CIFS], please check here.

[MS-DLTW] Microsoft Corporation, "Distributed Link Tracking: Workstation Protocol Specification", January 2007.

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

[MSDN-CJ] Microsoft Corporation, "Change Journals", http://msdn2.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

[REPARSE] Microsoft Corporation, "Reparse Points", http://msdn2.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://msdn2.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

1.3 Structure Overview (Synopsis)

This document describes the structure of common Windows native file system control codes (**FSCTL**), 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. Just as an application that needs to read from a file might use an operating system function to perform that read, an application that defragments a disk drive typically invokes the FSCTL_GET_RETRIEVAL_POINTERS and the FSCTL_MOVE_FILE file system controls through the operating system's device I/O control interface. 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 NT status 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

The Server Message Block (SMB) Protocol, as specified in [MS-SMB], relies 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 native Windows 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 Capability Negotiation

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.

2.1 Common Data Types

2.1.1 FILETIME Structure

The **FILETIME** structure is a 64-bit value that represents the number of 100-nanosecond intervals that have elapsed since January 1, 1601, in Coordinated Universal Time (UTC) format.

```
typedef struct _FILETIME {
  ULONG dwLowDateTime;
  ULONG dwHighDateTime;
} FILETIME;
```

dwLowDateTime: A 32-bit unsigned integer in little-endian format that contains the low-order bits of the file time.

dwHighDateTime: A 32-bit unsigned integer in little-endian format that contains the high-order bits of the file time.

2.1.2 REPARSE_DATA_BUFFER

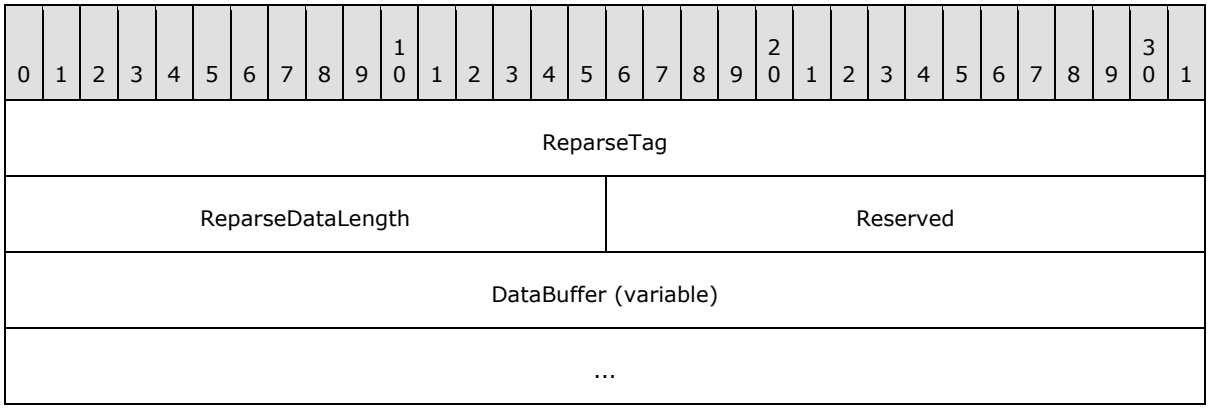
The REPARSE_DATA_BUFFER data element is used by the **NTFS** file system, and SMB/CIFS (Server Message Block/Common Internet File System), filter, and **minifilter** drivers that are implemented and shipped by Microsoft and the Windows I/O manager to store data for a reparse point. This data element (distinguished by a **ReparseTag** field whose high bit is set to 1) MUST be used only for reparse points that Windows components such as the I/O manager, the hierarchical storage manager filter, and the **Single-Instance Storage (SIS)** filter can interpret. This is so non-Microsoft software running on the Windows platform that makes use of reparse point data does not override the Microsoft interpretation of the data in those reparse points. Microsoft vendors MUST use a similar data structure for the **DataBuffer** member of the REPARSE_GUID_DATA_BUFFER structure. For more information about reparse points, see [REPARSE].

Each Windows component knows one or more reparse tag values that indicate a reparse point managed by that component. The server-side behavior associated with each type of reparse point is internal to Windows and does not affect the wire protocol. The specifications for the Symbolic Link Reparse Data Buffer and the Mount Point Reparse Data Buffer include the reparse tag value associated with each.

There is one generic type of REPARSE_DATA_BUFFER data element and two defined subtypes of the generic type for specific reparse point definitions. The filter driver associated with a reparse tag selects one of the formats from the following two sections that it will always use for the reparse data associated with that type of reparse point. The type of format selected is opaque to the NTFS file system.

2.1.2.1 Generic Reparse Data Buffer

The Generic Reparse Data Buffer data element is as follows:



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 reparse tag) of the reparse point.

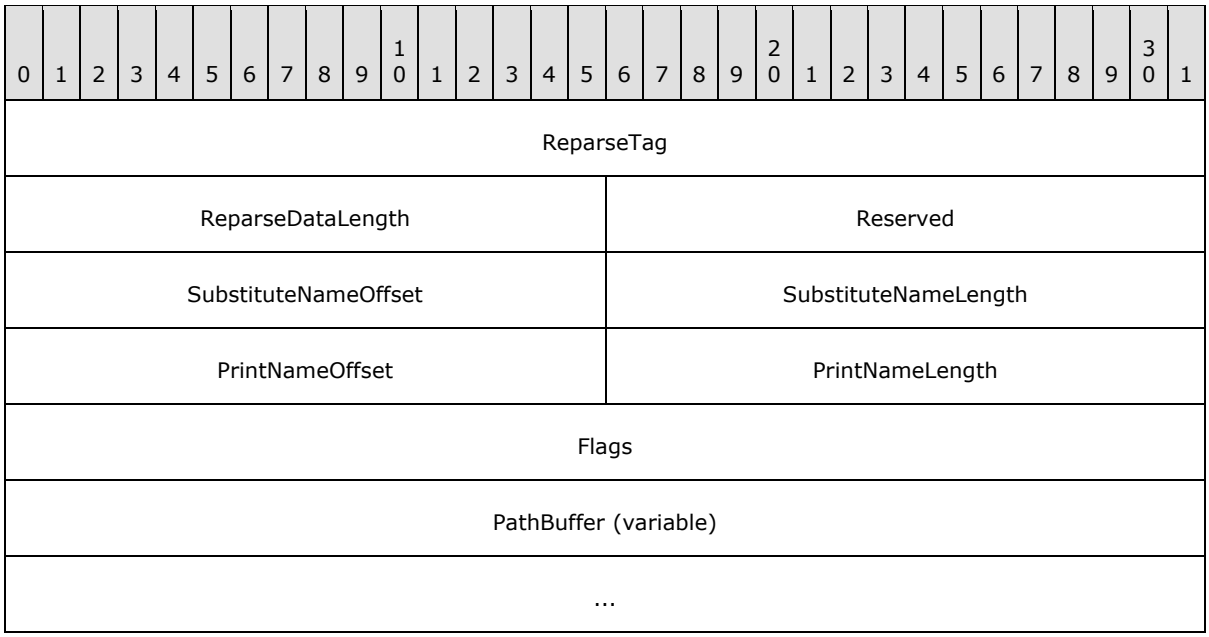
ReparseDataLength (2 bytes): A 16-bit unsigned integer value containing the size, in bytes, of the reparse data in the **DataBuffer** member.

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.

2.1.2.2 Symbolic Link Reparse Data Buffer

The Symbolic Link Reparse Data Buffer data element, which contains information on symbolic link reparse points, is as follows:



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, a reparse point tag assigned to Microsoft.

ReparseDataLength (2 bytes): A 16-bit unsigned integer value containing the size, in bytes, of the reparse data in the **PathBuffer** member. This value is the length of the data starting at the **SubstituteNameOffset** field (or SubstituteNameLength + PrintNameLength + 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 MUST contain 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 bit field that specifies whether the path name given in the **SubstituteName** field contains a full Windows NT path name or a path name relative to the source. A relative path name, such as "..\locals", accesses a directory relative to the current position in the directory hierarchy.

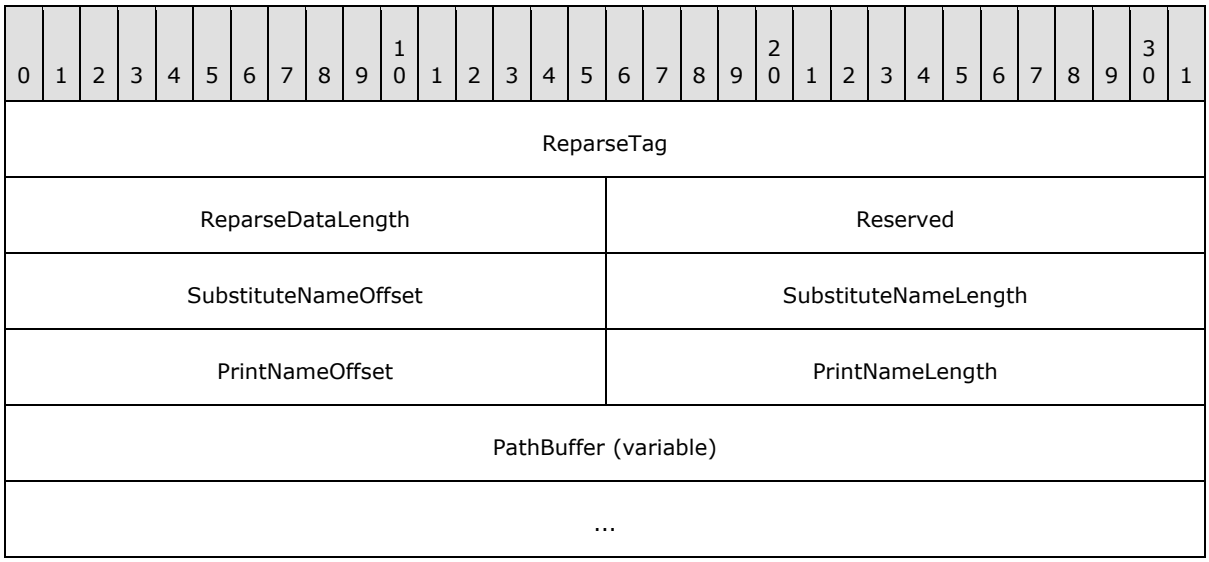
This field contains one of the values in the table below.

Value	Meaning
0x00000000	The path name given in the **SubstituteName** field contains a full Windows NT path name.
SYMLINK_FLAG_RELATIVE 0x00000001	When this **Flags** value is set, the given path name is relative to the source.

PathBuffer (variable): Unicode string 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.

2.1.2.3 Mount Point Reparse Data Buffer

The Mount Point Reparse Data Buffer data element, which contains information about mount point reparse points, is as follows.



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, a reparse point tag assigned to Microsoft.

ReparseDataLength (2 bytes): A 16-bit unsigned integer value containing the size, in bytes, of the reparse data in the **PathBuffer** member. This value is the length of the data starting at the **SubstituteNameOffset** field (or SubstituteNameLength + PrintNameLength + 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 MUST contain 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 string 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 REPARSE_GUID_DATA_BUFFER

The REPARSE_GUID_DATA_BUFFER data element MUST be used by all non-Microsoft file systems, filters, and minifilters to store data for a reparse point. Each reparse point contains one REPARSE_GUID_DATA_BUFFER structure.

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

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

If the high bit of the ReparseTag element is 0, an application MUST interpret the reparse point data as a REPARSE_GUID_DATA_BUFFER; otherwise, it MUST be interpreted as a REPARSE_DATA_BUFFER.

Each reparse point MUST contain one REPARSE_GUID_DATA_BUFFER structure. The REPARSE_GUID_DATA_BUFFER data element is as follows:

ReparseTag (4 bytes): A 32-bit unsigned integer value containing the reparse point tag that uniquely identifies the owner of the reparse point. This tag MUST match the reparse tag of the reparse point on which the FSCTL is being invoked.

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 undefined to the file system. On receipt, its content MUST be preserved and properly returned to the caller.

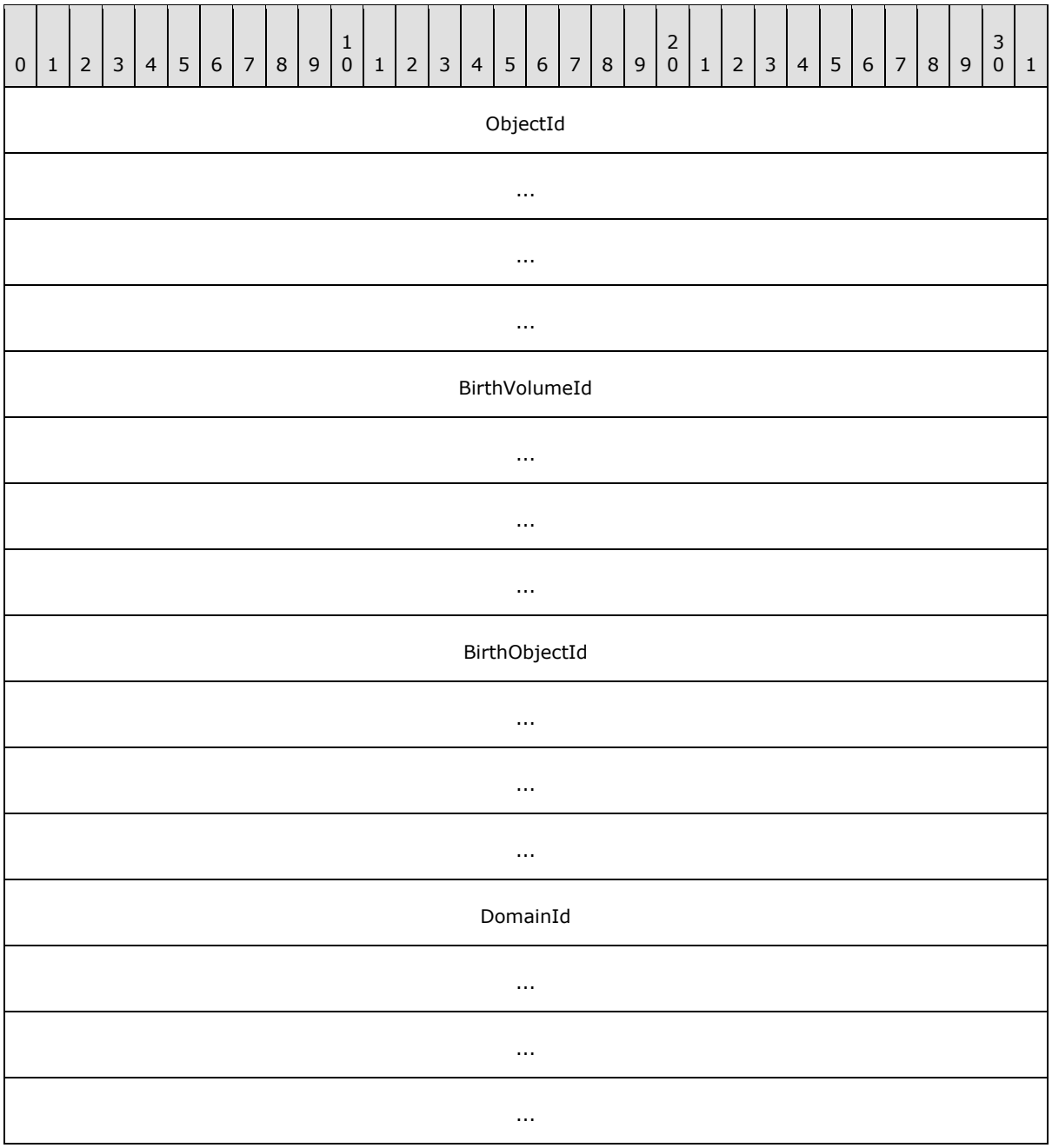
2.1.4 FILE_OBJECTID_BUFFER Structure

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

2.1.4.1 FILE_OBJECTID_BUFFER Type 1

The first possible structure for the FILE_OBJECTID_BUFFER data element follows:



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. Copy operations, move operations, or other file operations MAY change the value of the **ObjectId** member. Therefore, the BirthObjectId MAY not be the same as 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.<1>

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>

2.1.4.2 FILE_OBJECTID_BUFFER Type 2

The second possible structure for the FILE_OBJECTID_BUFFER data element 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.<3>

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 non-Microsoft RAID drivers or disk drivers) or above the file system (such as virus scanners).

An FSCTL MUST return a status of STATUS_INVALID_DEVICE_REQUEST when an FSCTL is not supported on the file system on which the file, directory, or volume handle specified by the FSCTL exists.<4>

2.3 FSCTL Structures

A process invokes an FSCTL on a handle to perform some 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. This section lists FSCTLs that are used across the wire by Windows.<5>

The table below specifies the FSCTL function numbers associated with the structures defined in section 2.1. Other FSCTL functions MAY be defined by the operating system; however, the FSCTL functions specified in this document are the only FSCTL functions that are permitted to be invoked across the network.

FSCTL name	FSCTL function number
FSCTL_ALLOW_EXTENDED_DASD_IO	0x90083
FSCTL_CREATE_OR_GET_OBJECT_ID	0x900c0
FSCTL_DELETE_OBJECT_ID	0x900a0
FSCTL_DELETE_REPARSE_POINT	0x900ac
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_GET_LINK_TRACKING_INFORMATION	0x1400e8
FSCTL_LMR_SET_LINK_TRACKING_INFORMATION	0x1400ec
FSCTL_PIPE_WAIT	0x110018
FSCTL_QUERY_ALLOCATED_RANGES	0x940cf
FSCTL_READ_FILE_USN_DATA	0x900eb
FSCTL_RECALL_FILE	0x90117
FSCTL_SET_COMPRESSION	0x9c040
FSCTL_SET_OBJECT_ID	0x90098
FSCTL_SET_OBJECT_ID_EXTENDED	0x900bc

FSCTL name	FSCTL function number
FSCTL_SET_REPARSE_POINT	0x900a4
FSCTL_SET_SPARSE	0x900c4
FSCTL_SET_ZERO_DATA	0x980c8
FSCTL_SIS_COPYFILE	0x90100
FSCTL_WRITE_USN_CLOSE_RECORD	0x900ef

2.3.1 FSCTL_ALLOW_EXTENDED_DASD_IO Request

This message instructs the server file system driver to not perform any I/O boundary checks on read or write calls performed on a **Direct Access to Storage Device (DASD) handle**. Instead, boundary checks are to be performed by the device driver software that communicates with the hardware on which the I/O is to be performed. Utility programs that want to read and write the disk directly, bypassing the file system, will open a DASD handle (that is, a handle to a volume) rather than a file handle. This message indicates that NTFSshould not check to see if reads or writes are going past the end of the volume and potentially on to another **partition** on the same drive.

This message does not contain any additional data elements.

The only data item this message returns is a status code, as specified in [MS-ERREF] section 4.

2.3.2 FSCTL_ALLOW_EXTENDED_DASD_IO Reply

This message returns the results of the FSCTL_ALLOW_EXTENDED_DASD_IO request.

The only data item this message returns is a status code, as specified in [MS-ERREF] section 4. 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 caller did not have sufficient permission to perform extended DASD I/O.<6>

2.3.3 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. The Win32 API for sending an FSCTL (DeviceIoControl) takes a file handle as a parameter. This file handle specifies the file whose object ID is queried or set.

This message does not contain any additional data elements.

2.3.4 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 2.1.4).

This message also returns a status code, as specified in [MS-ERREF] section 4. 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 NTFS is unable to generate a unique (to this volume) ID.<7>
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.

2.3.5 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.

2.3.6 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 4. 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 is not large enough to contain a FILE_OBJECTID_BUFFER structure.
STATUS_ACCESS_DENIED 0xC0000022	The handle was not opened with restore access or write access.
STATUS_OBJECT_NAME_NOT_FOUND 0xC0000034	The file or directory has no object ID.

2.3.7 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 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 interprets 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 is because a particular filter driver SHOULD be implemented with specific support for the type of reparse point data structure it accepts. A REPARSE_DATA_BUFFER structure is used by Microsoft filters, and MUST NOT be used by non-Microsoft filters; a REPARSE_GUID_DATA_BUFFER can be used by non-Microsoft or Microsoft

filters. The protocol itself MUST NOT differentiate between the two types of reparse point data structures.

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

2.3.8 FSCTL_DELETE_REPARSE_POINT Reply

This message returns the result of the FSCTL_DELETE_REPARSE_POINT request.

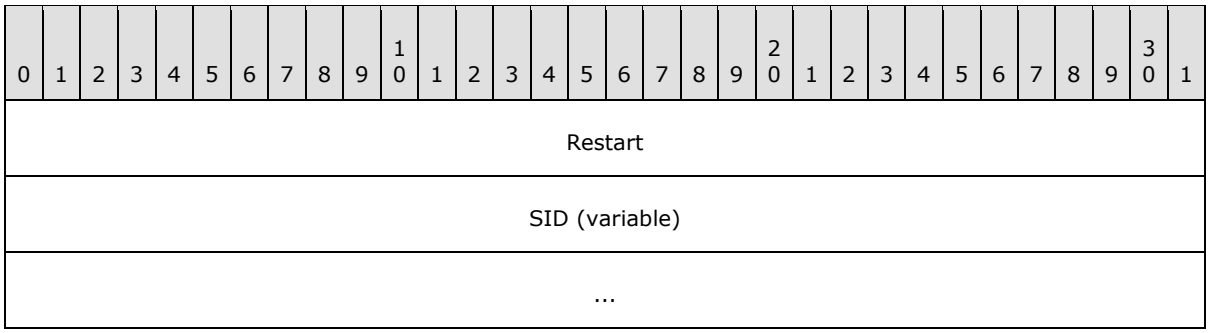
The only data item this message returns is a status code, as specified in [MS-ERREF] section 4. The status code returned directly by the function that processes this FSCTL MUST be STATUS_SUCCESS or one of the following:

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_INVALID_BUFFER_SIZE 0xC0000206	The input buffer is null.
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 non-Microsoft 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 non-Microsoft tag, and the Reparse GUID provided does not match the one in the reparse point for this file or directory.

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 (in the directory associated with the handle on which this FSCTL was invoked) whose owner matches the specified **security identifier (SID)**. This message contains a FIND_BY_SID_DATA data element.

The FIND_BY_SID_DATA data element is as follows:



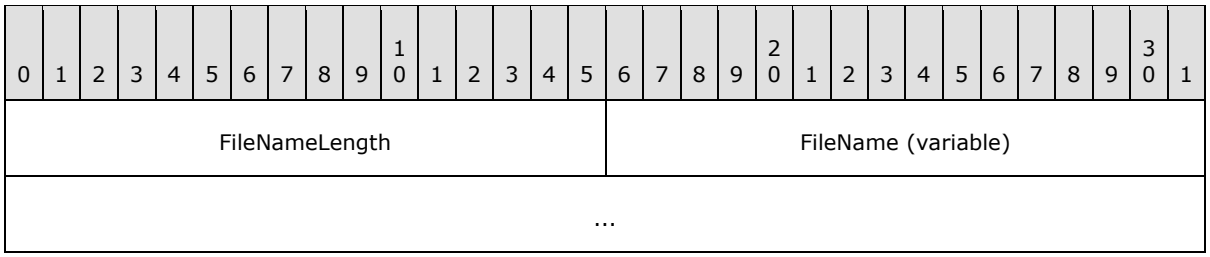
Restart (4 bytes): A 32-bit unsigned integer value that indicates to restart the search. This value MUST be 1 on first call so that the search starts from the root. For subsequent calls, this member SHOULD be zero so that the search resumes at the point where it stopped.

SID (variable): A SID (security identifier) (see [MS-SECO]) 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 as an array of FIND_BY_SID_OUTPUT data elements, one for each matching file that is found.

The FIND_BY_SID_OUTPUT data element is as follows:



FileNameLength (2 bytes): A 16-bit unsigned integer value containing the size of the file name in bytes. This size does not include the NULL character.

FileName (variable): A null-terminated Unicode string that specifies the fully qualified path name for the file.

This message also returns a status code, as specified in [MS-ERREF] section 4. 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 the handle to a 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 a 4-byte boundary, or the output buffer is not large enough to hold the file name length and file name that is returned by this

Error code	Meaning
	message, or the restart value is greater than 1.

2.3.11 FSCTL_GET_COMPRESSION Request

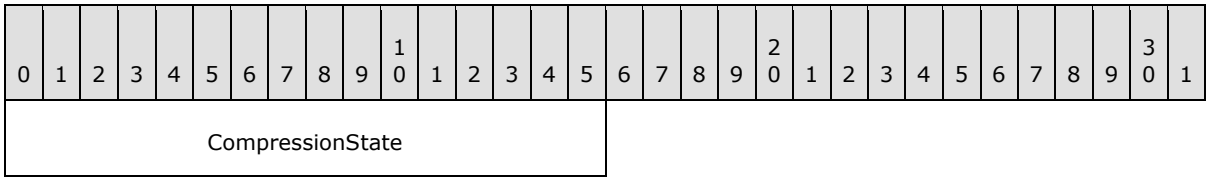
This message requests that the server return the current compression state of the file or directory associated with the handle (on which this FSCTL was invoked) on a volume whose file system supports per-**stream** compression.

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_DEFAULT 0x0001	The file or directory is compressed by using the default compression algorithm.<8>
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 by Microsoft and SHOULD NOT be used by others.

The actual file or directory compression that is performed when a server receives a request for COMPRESSION_FORMAT_DEFAULT and COMPRESSION_FORMAT_LZNT1 is implementation-dependent.<9>

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.

The only data item that this message returns is a status code, as specified in [MS-ERREF] section 4. 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.<10>

2.3.13 FSCTL_GET_NTFS_VOLUME_DATA Request

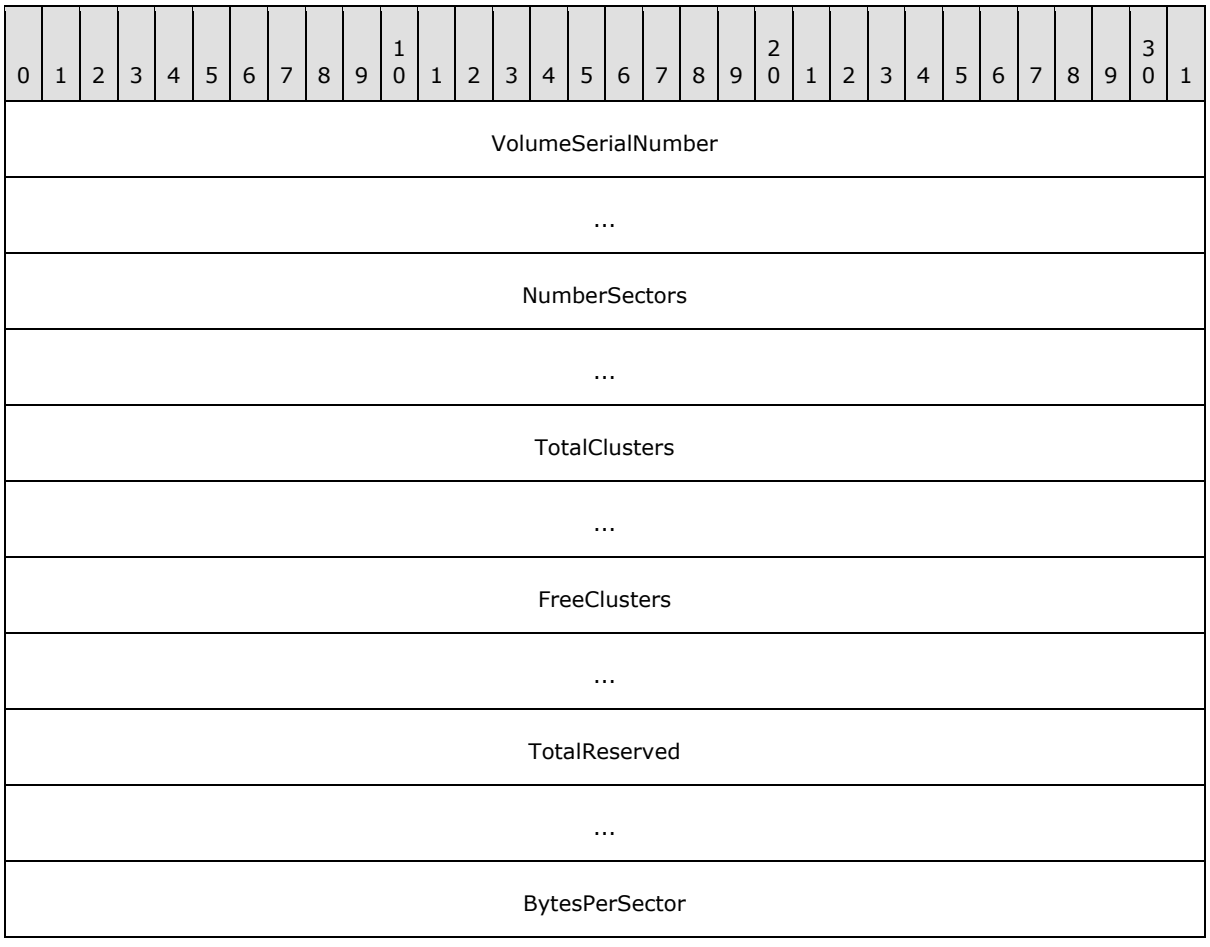
This message requests that the server return information on the specified NTFS file system volume.

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_REPLY element.

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



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** 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.

This message also returns a status code, as specified in [MS-ERREF] section 4. 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.

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. Modifying an object identifier can result in loss of data from portions of a file up to and including entire volumes of data.

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 2.1.4).

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 4. 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 object ID is NULL, 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 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 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 interprets 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. A REPARSE_DATA_BUFFER structure is used by Microsoft filters, and MUST NOT be used by non-Microsoft filters; a REPARSE_GUID_DATA_BUFFER can be used by non-Microsoft and Microsoft filters.

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.<11>

This message returns a status code, as specified in [MS-ERREF] section 4. The status code returned directly by the function that processes this FSCTL MUST be STATUS_SUCCESS or one of the following:

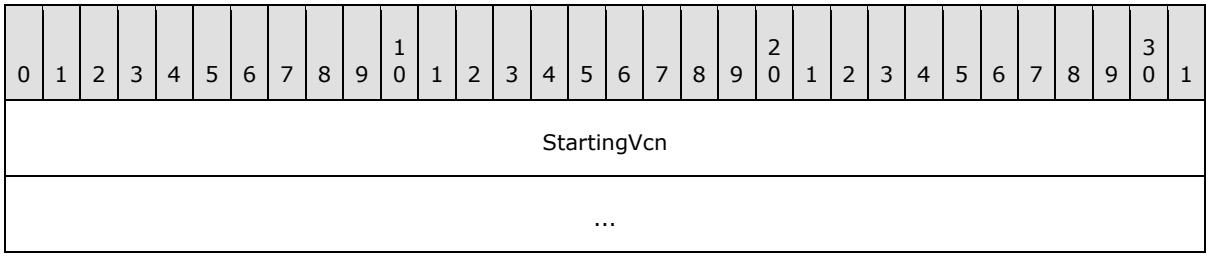
Error code	Meaning
STATUS_BUFFER_TOO_SMALL 0xC0000023	The output buffer is too small to contain a REPARSE_GUID_DATA_BUFFER.
STATUS_INVALID_PARAMETER 0xC000000D	The handle is not to a file or directory.

Error code	Meaning
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 variably sized data element, StartingVcn, that describes the location on disk of the file or directory associated with the handle on which this FSCTL was invoked. It is most commonly used by defragmentation utilities. This data element describes the mapping between **virtual cluster numbers** and logical cluster numbers.

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 MAY be rounded down to the first VCN of the extent in which the given extent is found. 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.

For the NTFS file system, 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.<12>

The RETRIEVAL_POINTERS_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
ExtentCount																															
Unused																															
StartingVcn																															
...																															
Extents (variable)																															
...																															

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): Unused area for data alignment purposes.

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 round down to the first VCN of the extent in which the requested starting VCN is found. 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:

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
NextVcn																															
...																															
Lcn																															
...																															

NextVcn (8 bytes): A 64-bit unsigned 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. The length is an input to the FSCTL_MOVE_FILE request.

Lcn (8 bytes): A 64-bit unsigned integer that contains the logical cluster number (LCN) at which the current extent begins on the volume. On the NTFS file system, a 64-bit value of 0xFFFFFFFFFFFFFFFF 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]. NTFS performs compression 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.

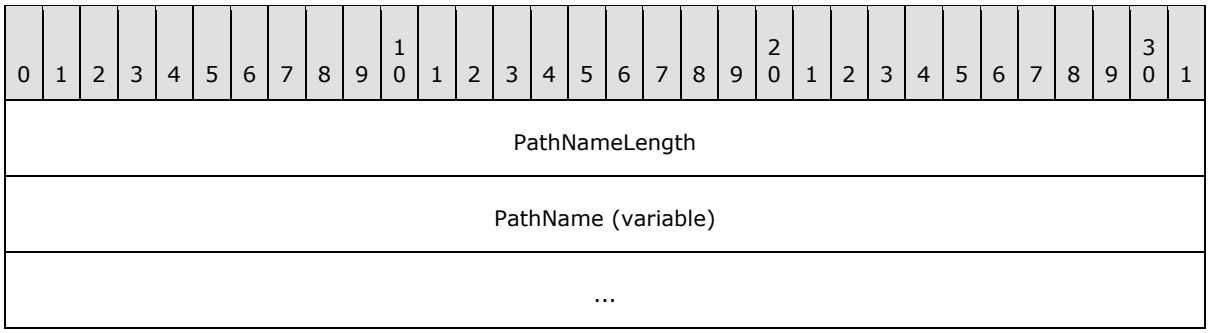
This message returns a status code, as specified in [MS-ERREF] section 4. The status code returned directly by the function that processes this FSCTL MUST be STATUS_SUCCESS or one of the following:

Error code	Meaning
STATUS_BUFFER_TOO_SMALL 0xC0000023L	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 return whether or not the specified path name associated with the handle on which this FSCTL was invoked is composed of valid characters with respect to the remote file system storage.

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.

The only data item this message returns is a status code, as specified in [MS-ERREF] section 4. If the request is successful, the status code returned by the server MUST have bit 31 (0xC0000000) clear. Otherwise, a status code indicating an error, informational status, or warning status MUST be returned, as specified in [MS-ERREF] section 4.<13>

2.3.23 FSCTL_LMR_GET_LINK_TRACKING_INFORMATION Request

This message requests **Distributed Link Tracking** information (see [MS-DLTW] section 3.1.6) such as file system type and volume ID, as specified in section 2.3.24, for the file associated with the handle on which this FSCTL was invoked.

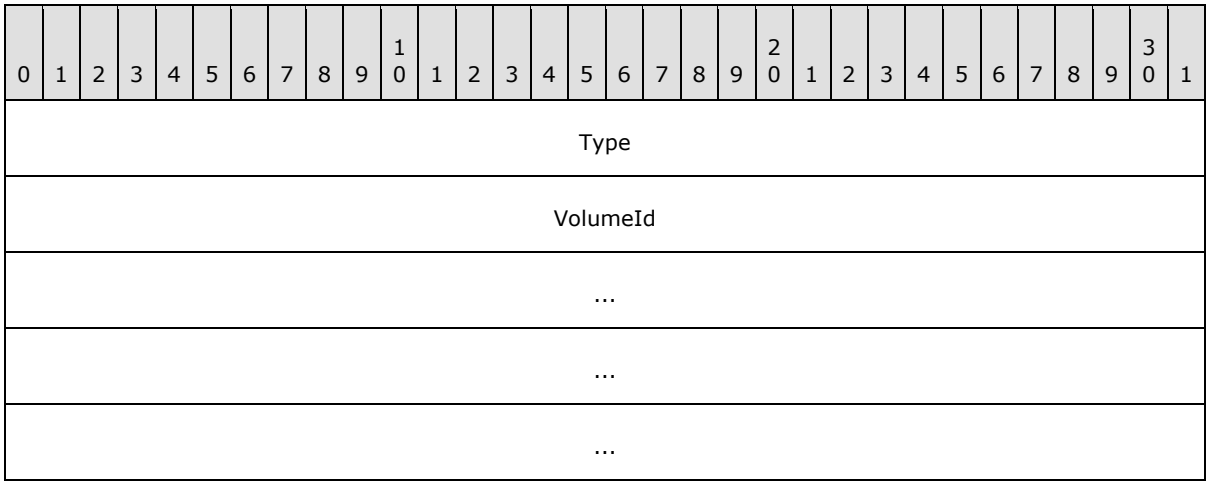
This message MUST only be sent for a file handle.

This message does not contain any additional data elements.

2.3.24 FSCTL_LMR_GET_LINK_TRACKING_INFORMATION Reply

The FSCTL_LMR_GET_LINK_TRACKING_INFORMATION reply message returns the results of the FSCTL_LMR_GET_LINK_TRACKING_INFORMATION request.

The LINK_TRACKING_INFORMATION data element is 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. This value 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 reply to an FSCTL_LMR_GET_LINK_TRACKING_INFORMATION request, called by using the file handle of the destination file.

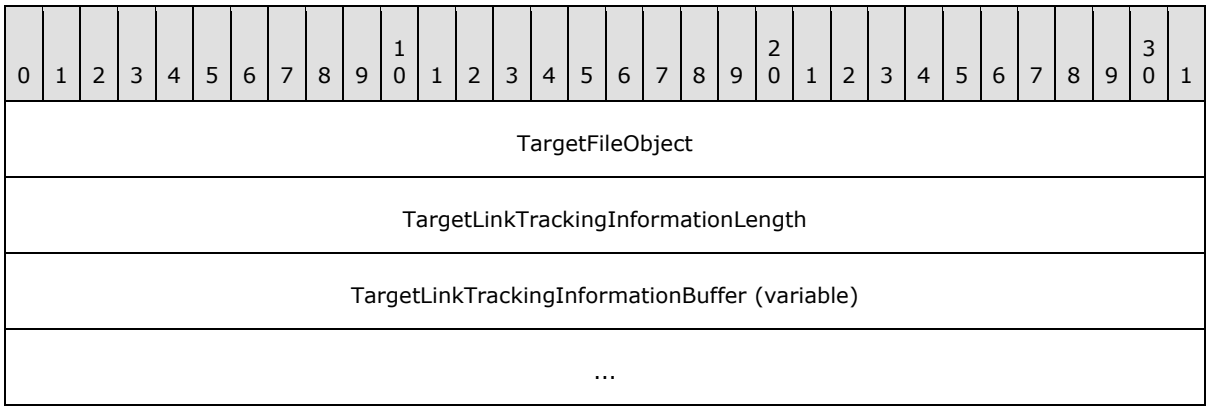
This message also returns a status code, as specified in [MS-ERREF] section 4. The status code returned directly by the function that processes this FSCTL MUST be STATUS_SUCCESS or one of the following:

Error code	Meaning
STATUS_INVALID_PARAMETER 0xC000000D	The output buffer is too small to contain a LINK_TRACKING_INFORMATION structure, or no link tracking information exists for this file or directory.

2.3.25 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 associated with the handle on which this FSCTL was invoked. The message contains a REMOTE_LINK_TRACKING_INFORMATION data element. For more information about Distributed Link Tracking, see [MS-DLTW] section 3.1.6.

The REMOTE_LINK_TRACKING_INFORMATION data element is as follows:



TargetFileObject (4 bytes): Fid of the file from which to obtain link tracking information. For Fid type, see [MS-SMB] section 2.2.14.7.

TargetLinkTrackingInformationLength (4 bytes): Length of the **TargetLinkTrackingInformationBuffer**.

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

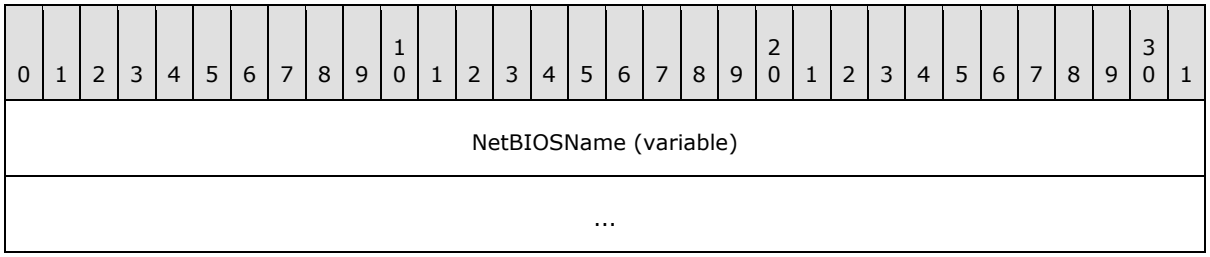
2.3.25.1 TARGET_LINK_TRACKING_INFORMATION_Buffer

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.

2.3.25.1.1 TARGET_LINK_TRACKING_INFORMATION_Buffer_1

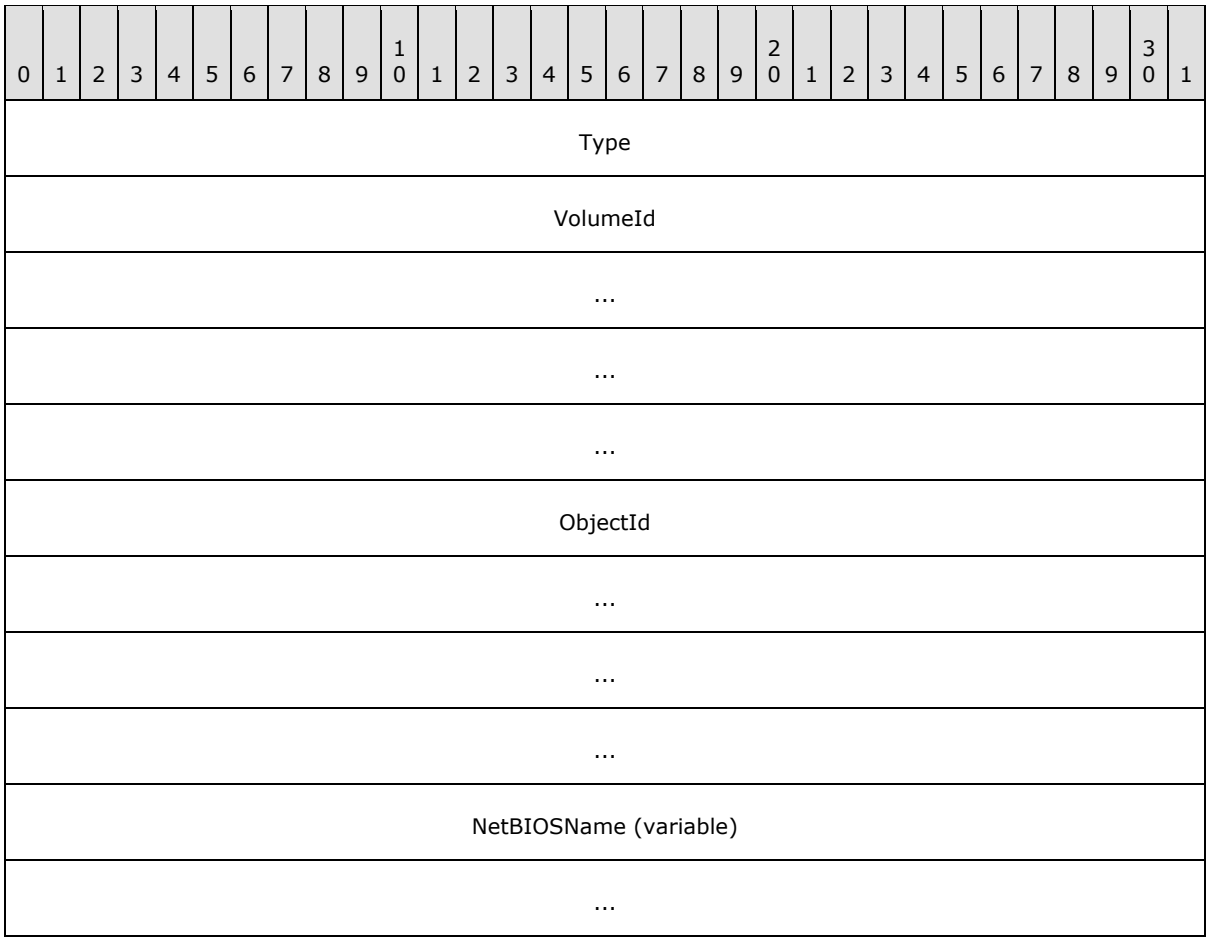
If the **TargetLinkTrackingInformationLength** value is less than 36, the TARGET_LINK_TRACKING_INFORMATION_Buffer data element MUST be as follows:



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.

2.3.25.1.2 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 reply to an FSCTL_LMR_GET_LINK_TRACKING_INFORMATION request, called using the file handle of the destination file.

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.

2.3.26 FSCTL_LMR_SET_LINK_TRACKING_INFORMATION Reply

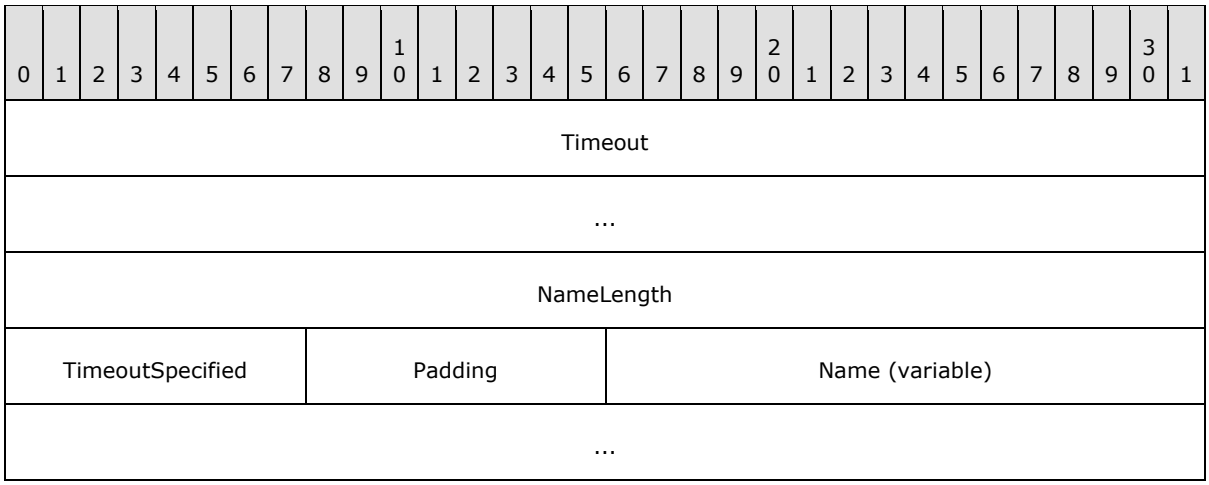
This message returns the results of the FSCTL_LMR_SET_LINK_TRACKING_INFORMATION request.

The only data item this message returns is a status code, as specified in [MS-ERREF] section 4. 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 0.

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.

A positive value expresses an absolute time at which the base time is the beginning of the year 1601 A.D. in the Gregorian calendar. A negative value expresses a time interval relative to some base time, typically the current time.

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

TimeoutSpecified (1 byte): An 8-bit unsigned value that specifies whether or not the **Timeout** parameter will be ignored.

Value	Meaning
FALSE 0	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.

Value	Meaning
1	

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".

2.3.28 FSCTL_PIPE_WAIT Reply

This message returns the results of the FSCTL_PIPE_WAIT request.

If an instance of the pipe is available before the time-out interval elapses, the return value is nonzero.

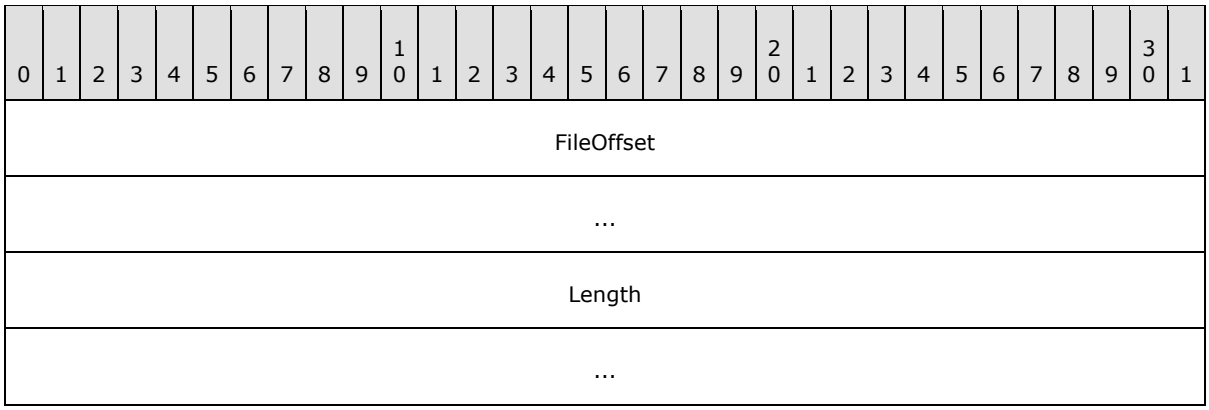
If an instance of the pipe is not available before the time-out interval elapses, the return value is zero.

If no instances of the specified named pipe exist, the FSCTL_PIPE_WAIT reply returns immediately, regardless of the time-out value.

2.3.29 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 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.

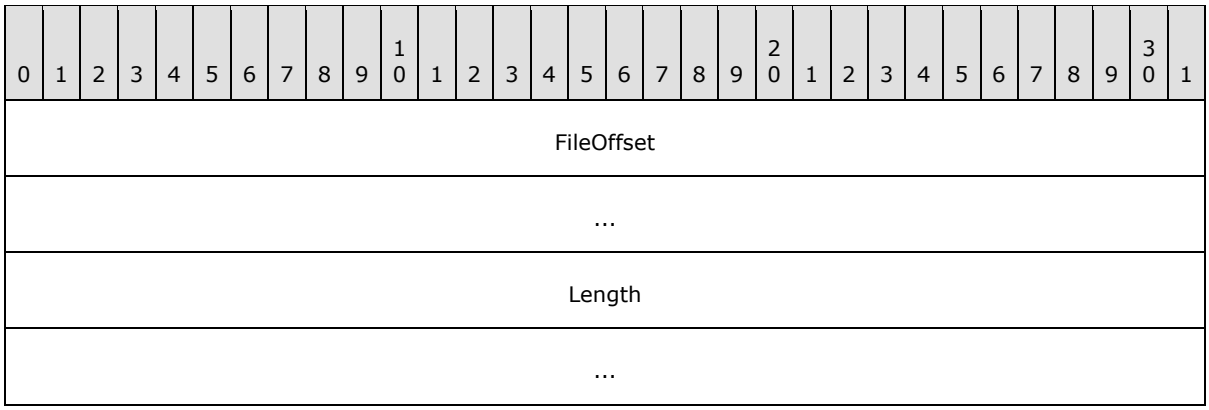
Length (8 bytes): A 64-bit signed integer that contains the size, in bytes, of the range. The value of this field MUST be greater than or equal to 0.

2.3.30 FSCTL_QUERY_ALLOCATED_RANGES Reply

The FSCTL_QUERY_ALLOCATED_RANGES reply message returns the results of the FSCTL_QUERY_ALLOCATED_RANGES request.

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 SMB, the lower-level protocol that carries the FSCTL) by the size of the FILE_ALLOCATED_RANGE_BUFFER element. Ranges returned are always at least partially within 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.<14>

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.<15> 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.<16>

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

This message returns a status code, as specified in [MS-ERREF] section 4. 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 is less than 0, or the given Length is less than 0, or the given FileOffset plus the given Length is larger than 0x7FFFFFFFFFFFFFFF.
STATUS_INVALID_USER_BUFFER 0xC00000E8	The input buffer or output buffer is not aligned to a 4-byte boundary.

Error code	Meaning
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.

2.3.31 FSCTL_READ_FILE_USN_DATA Request

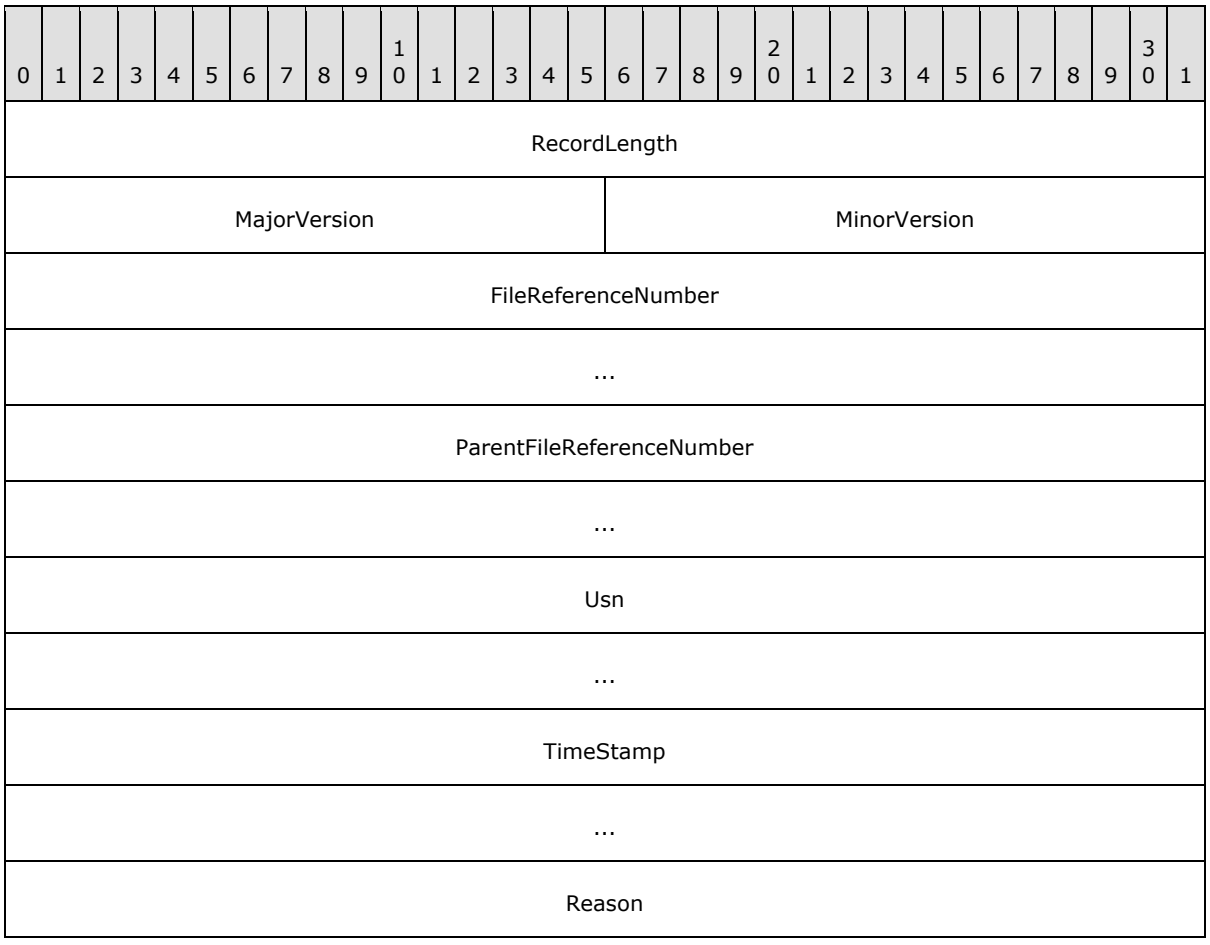
This message requests that the server return the change journal information for the file or directory associated with the handle on which this FSCTL was invoked.

This message does not contain any additional data elements.

2.3.32 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:



SourceInfo	
SecurityId	
FileAttributes	
FileNameLength	FileNameOffset
FileName (variable)	
...	

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.<17>

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).<18>

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.<19>

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 (update sequence number) 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. For more information, see [MSDN-CJ].

TimeStamp (8 bytes): A structure containing the absolute system time in UTC expressed as the number of 100-nanosecond intervals since January 1, 1601 (UTC), in the format of a **FILETIME (section 2.1.1)** structure.

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 by Microsoft and SHOULD NOT be used by others).

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.
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 Windows–based 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	An NTFS file system hard link is added to (or removed from) the file or directory. An NTFS file system hard link, similar to a POSIX hard link, is one of several directory entries that see the same 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.

Value	Meaning
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 (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 a unique security identifier (SID) assigned to the file or directory associated with this record.

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 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.

This message returns a status code, as specified in [MS-ERREF] section 4. 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.

2.3.33 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. An application SHOULD not recall files that it does not touch for performance reasons.

This message does not contain any additional data elements.

2.3.34 FSCTL_RECALL_FILE Reply

This message returns the results of the FSCTL_RECALL_FILE request.

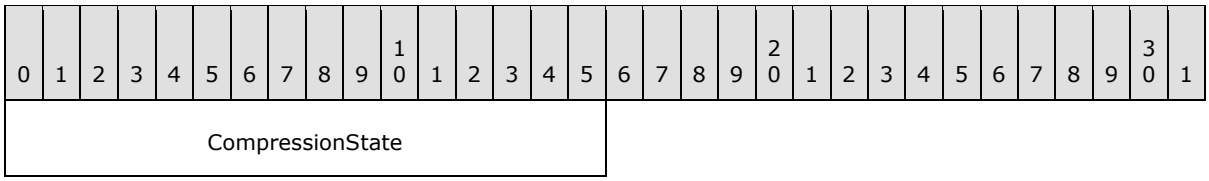
The only data item this message returns is a status code, as specified in [MS-ERREF] section 4. The status code returned directly by the function that processes this FSCTL MUST be STATUS_SUCCESS or one of the following:

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

2.3.35 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) on a volume whose file system supports per-stream compression. 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.<20>
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 by Microsoft, and SHOULD NOT be used by others. This does not restrict the use of alternative compression algorithms by others. Because compressed data does not travel across the wire in the course of FSCTL, FileInformation class, or FileSystemInformation class requests or replies, an implementation can associate any local compression mechanisms with the above values. Particularly, this specification does not require that a non-Microsoft implementation implement the LZNT1 compression algorithm for its file systems for interoperability.

The actual file or directory compression performed when a server receives a request for COMPRESSION_FORMAT_DEFAULT and COMPRESSION_FORMAT_LZNT1 is implementation-dependent.<21>

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

2.3.36 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 4. 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 less than 2, or the handle is not to a file or directory, or the requested CompressionState is not 1 or 2, or the file or directory is encrypted.
STATUS_INVALID_DEVICE_REQUEST 0xC0000010	The volume does not allow compression.
STATUS_DISK_FULL 0xC000007	The disk is full.

2.3.37 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 2.1.4) data element.<22><23>

2.3.38 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.

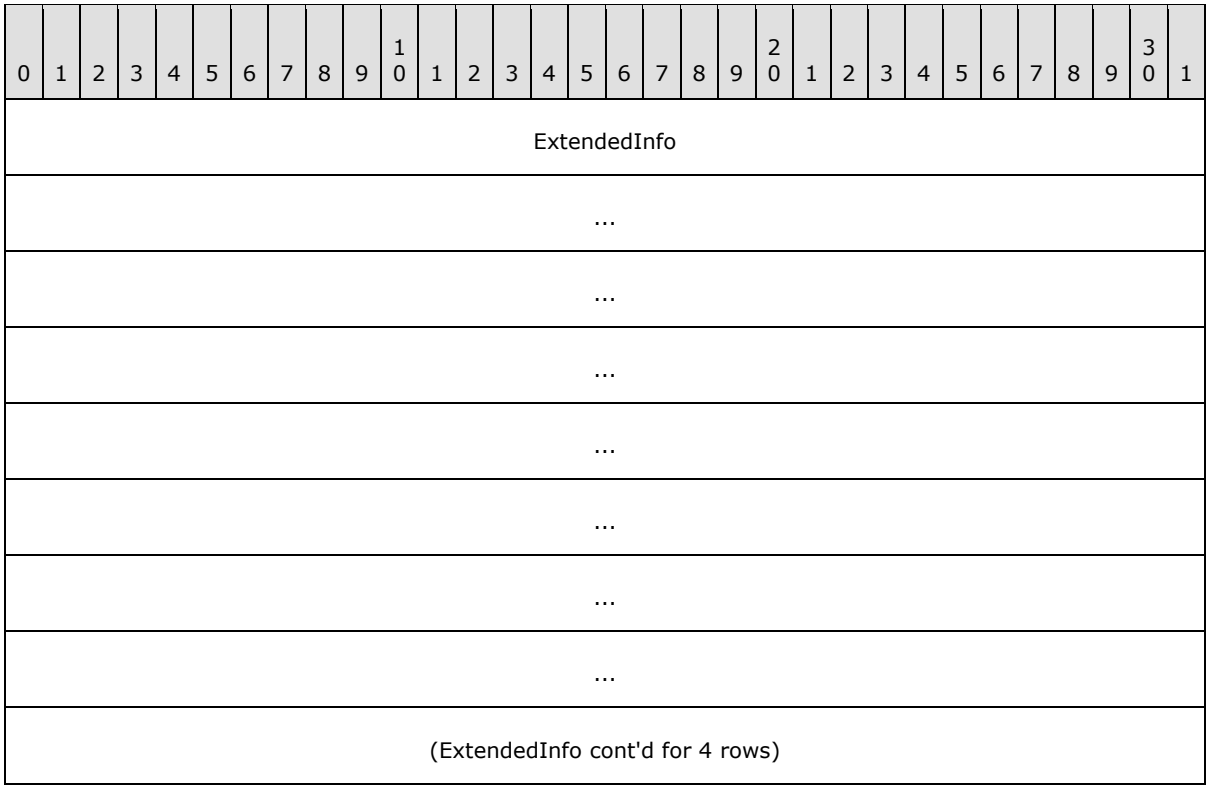
The only data item this message returns is a status code, as specified in [MS-ERREF] section 4. The status code returned directly by the function that processes this FSCTL MUST be STATUS_SUCCESS or one of the following:

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 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.

2.3.39 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:



ExtendedInfo (48 bytes): A 48-byte BLOB containing user-defined extended data that was passed to this FSCTL by an application. In this situation, the user refers to the developer 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.<24>

2.3.40 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 4. 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.

2.3.41 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 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 interprets 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.

2.3.42 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 4. 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_INVALID_BUFFER_SIZE 0xC0000206	The input buffer is NULL.
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 non-Microsoft reparse tag, or the GUID specified for a non-Microsoft reparse tag does not match the GUID known by the operating system for this reparse point, or the reparse tag is 0 or 1.

Error code	Meaning
STATUS_INVALID_DEVICE_REQUEST 0xC0000010	The file system does not support reparse points.

2.3.43 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 needed as the file is written.

This message does not contain any additional data elements.

2.3.44 FSCTL_SET_SPARSE Reply

This message returns the results of the FSCTL_SET_SPARSE request.

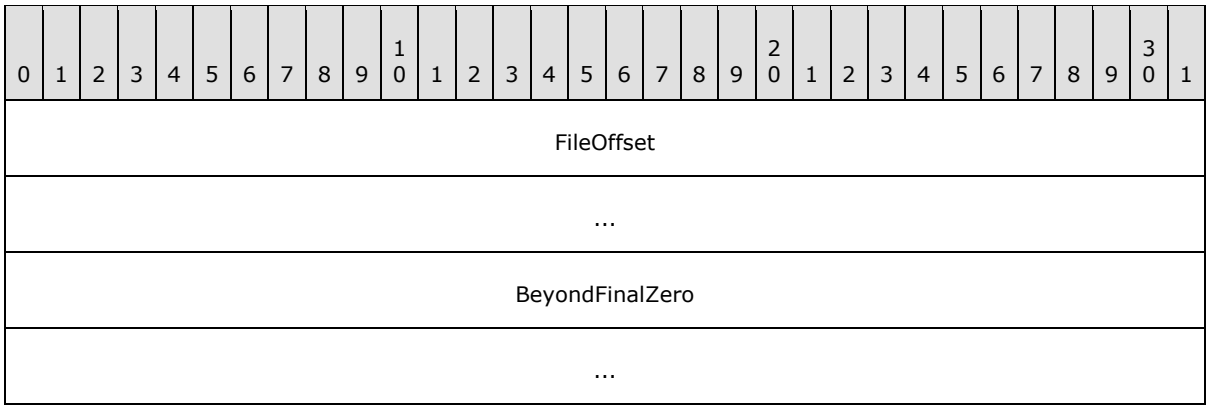
The only data item this message returns is a status code, as specified in [MS-ERREF] section 4. The status code returned directly by the function that processes this FSCTL MUST be STATUS_SUCCESS or one of the following:

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.

2.3.45 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.<25>

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

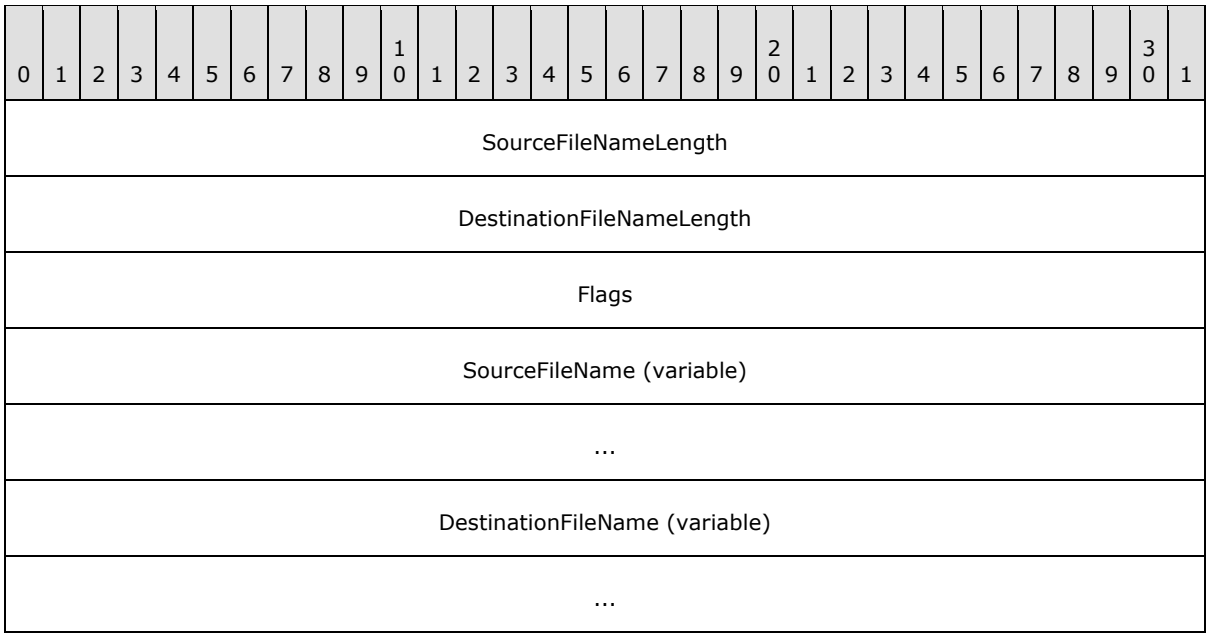
Error code	Meaning
STATUS_INVALID_PARAMETER 0xC000000D	The handle is not to a file, or input buffer length is not equal to the size of a FILE_ZERO_DATA_INFORMATION structure, or the given **FileOffset** is less than zero, or the given **BeyondFinalZero** is less than zero, or the given **FileOffset**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

2.3.47 FSCTL_SIS_COPYFILE Request

The FSCTL_SIS_COPYFILE request message requests that the server use the Single-Instance Storage (SIS) filter to copy the file that is associated with the handle on which this FSCTL was invoked. 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 source file to the 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.

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 below 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.<26>

2.3.48 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 4. 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. 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_OBJECT_NAME_COLLISION 0xC0000035	The COPYFILE_SIS_REPLACE flag was not specified, and the destination file exists.
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.
STATUS_INVALID_DEVICE_REQUEST 0xC0000010	The Single-Instance Storage (SIS) filter is not installed on the server.<27>

2.3.49 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 post any pending USN updates for the indicated file.

2.3.50 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.

The only data item this message returns is a status code, as specified in [MS-ERREF] section 4. 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.

Error code	Meaning
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 table below) that specify what information on a file is to be queried. Some of the file information classes 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.<28>

File information class	Level
FileAccessInformation	8
FileAlignmentInformation	17
FileAllInformation	18
FileAllocationInformation	19
FileAlternateNameInformation	21
FileAttributeTagInformation	35
FileBasicInformation	4
FileBothDirectoryInformation	3
FileCompletionInformation	30
FileCompressionInformation	28
FileDirectoryInformation	1
FileDispositionInformation	13
FileEaInformation	7
FileEndOfFileInformation	20
FileFullDirectoryInformation	2
FileFullEaInformation	15
FileHardLinkInformation	46
FileIdBothDirectoryInformation	37
FileIdFullDirectoryInformation	38
FileInternalInformation	6
FileLinkInformation	11
FileMailslotQueryInformation	26

File information class	Level
FileMailslotSetInformation	27
FileModeInformation	16
FileMoveClusterInformation	31
FileNameInformation	9
FileNamesInformation	12
FileNetworkOpenInformation	34
FileObjectIdInformation	29
FilePipeInformation	23
FilePipeLocalInformation	24
FilePipeRemoteInformation	25
FilePositionInformation	14
FileQuotaInformation	32
FileRenameInformation	10
FileReparsePointInformation	33
FileShortNameInformation	40
FileStandardInformation	5
FileStreamInformation	22
FileTrackingInformation	36
FileValidDataLengthInformation	39

2.4.1 FileAccessInformation

This information class is used to query or set the access rights of the file.

The FILE_ACCESS_INFORMATION data element is as follows:

AccessFlags (4 bytes): A **DWORD** that MUST contain values specified in **ACCESS_MASK** of [MS-DTYP].

2.4.2 FileAllInformation

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

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 2.4.36.

InternalInformation (8 bytes): A FILE_INTERNAL_INFORMATION structure specified in section 2.4.19.

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 2.4.31.

ModeInformation (4 bytes): A FILE_MODE_INFORMATION structure specified in section 2.4.23.

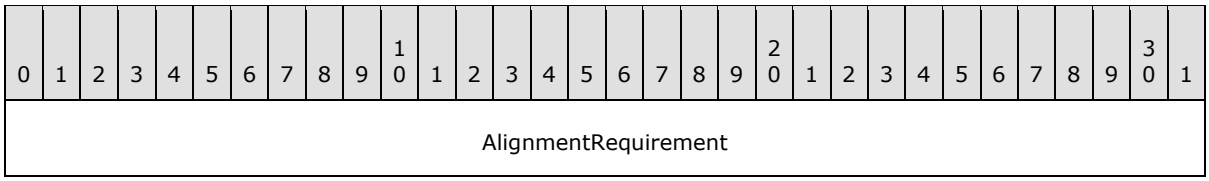
AlignmentInformation (4 bytes): A FILE_ALIGNMENT_INFORMATION structure specified in section 2.4.3.

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

2.4.3 FileAlignmentInformation

The buffer alignment required by the underlying device.

The FILE_ALIGNMENT_INFORMATION data element is as follows.



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.
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.

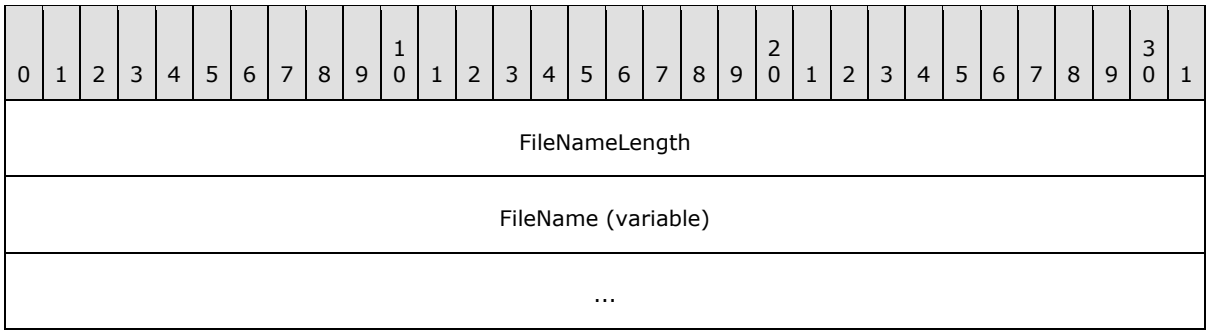
2.4.4 FileAllocationInformation

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. Usually, this value is a multiple of the sector or cluster size of the underlying physical device. This value MUST be greater than or equal to 0.<29>

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 (8 characters that appear before the "." and 3 characters that appear after). This name exists for compatibility with MS-DOS and 16-bit versions of Windows, which did not support longer file names or spaces within file names. A file MAY have an alternate name if its full name is not compliant with the restrictions for file names under MS-DOS and 16-bit Windows.<30>

The FILE_NAME_INFORMATION data element is as follows:



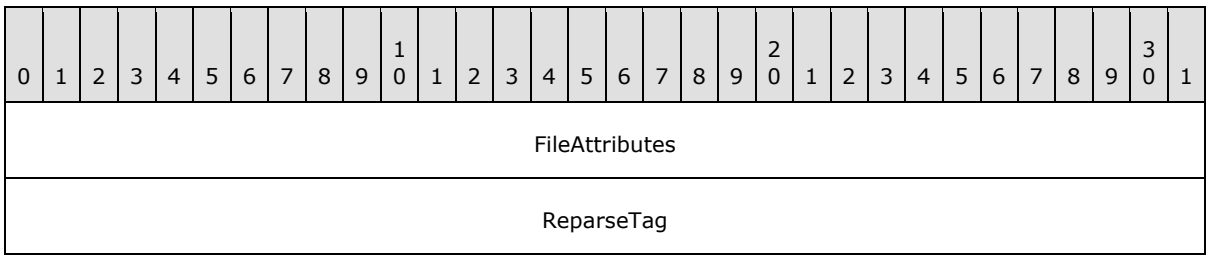
FileNameLength (4 bytes): A 32-bit unsigned integer that contains the length in bytes of the **FileName** member.

FileName (variable): A sequence of Unicode characters containing the file name. This field might not be NULL-terminated, and MUST be handled as a sequence of **FileNameLength** bytes.

2.4.6 FileAttributeTagInformation

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

The FILE_ATTRIBUTE_TAG_INFORMATION data element is as follows.



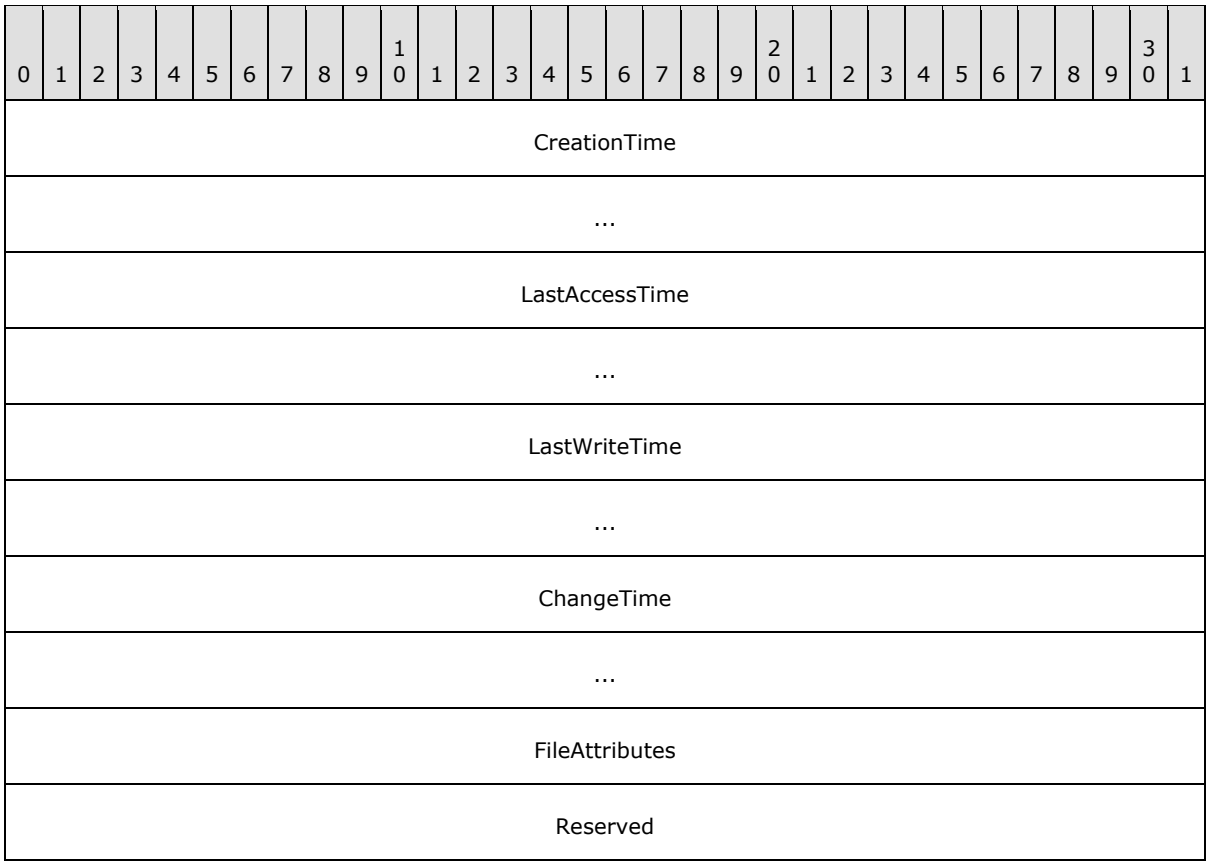
FileAttributes (4 bytes): A 32-bit unsigned integer that contains the file attributes. Valid file attributes are as specified in section 2.6.

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.

2.4.7 FileBasicInformation

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

The FILE_BASIC_INFORMATION data element is as follows:



CreationTime (8 bytes): A 64-bit signed integer that contains the time when the file was created. All dates and times in this message are in absolute system-time format, which is represented as a **FILETIME (section 2.1.1)** structure. This field should be set to an integer value greater than or equal to 0; alternately, it can be set to (-1) to indicate that this time field should not be updated by the server.

LastAccessTime (8 bytes): A 64-bit signed integer that contains the last time the file was accessed in the format of a **FILETIME** structure. This field should be set to an integer value greater than or equal to 0; alternately, it can be set to (-1) to indicate that this time field should not be updated by the server.

LastWriteTime (8 bytes): A 64-bit signed integer that contains the last time information was written to the file in the format of a **FILETIME** structure. This field should be set to an integer value greater than or equal to 0; alternately, it can be set to (-1) to indicate that this time field should not be updated by the server.

ChangeTime (8 bytes): A 64-bit signed integer that contains the last time the file was changed in the format of a **FILETIME** structure. This field should be set to an integer value greater than or equal to 0; alternately, it can be set to (-1) to indicate that this time field should not be updated by the server.

FileAttributes (4 bytes): A 32-bit unsigned integer that contains the file attributes. Valid file attributes are specified in section 2.6.<31>

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

2.4.8 FileBothDirectoryInformation

This information class is used to query detailed information for the files in a directory.

The FILE_BOTH_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																															

EaSize		
ShortNameLength	Reserved	ShortName
...		
...		
...		
...		
...		
...		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), and MUST NOT assume that the value of NextEntryOffset is the same as the size of the current entry.

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.<32>

CreationTime (8 bytes): A 64-bit signed integer that contains the time when the file was created. All dates and times are in absolute system-time format, which is represented as a **FILETIME (section 2.1.1)** structure. This value MUST be greater than or equal to 0.

LastAccessTime (8 bytes): A 64-bit signed integer that contains the last time the file was accessed in the format of a **FILETIME** structure. This value MUST be greater than or equal to 0.

LastWriteTime (8 bytes): A 64-bit signed integer that contains the last time information was written to the file in the format of a **FILETIME** structure. This value MUST be greater than or equal to 0.

ChangeTime (8 bytes): A 64-bit signed integer that contains the last time the file was changed in the format of a **FILETIME** structure. 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. Usually, this value is a multiple of the sector or cluster size of the underlying physical device. 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 file attributes are specified in section 2.6.

FileNameLength (4 bytes): A 32-bit unsigned integer that contains the length, in bytes, of the **FileName** field. The NULL termination of the string, if present, is not included in the FileNameLength count.

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): MUST be an 8-bit signed value containing the length, in bytes, of the **ShortName** field. If there is a NULL termination at the end of the string, it is not included in the **FileNameLength** count. This value MUST be greater than or equal to 0.

Reserved (1 byte): Reserved for alignment.

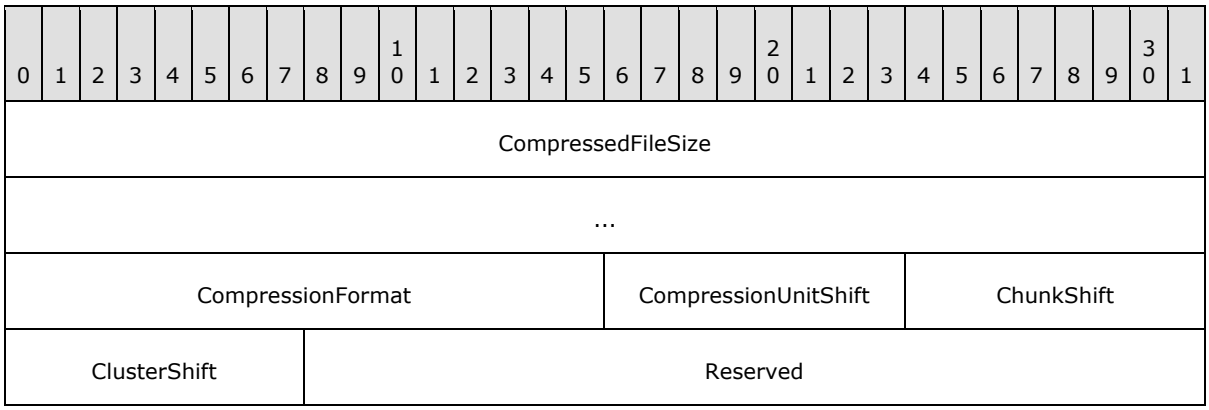
ShortName (24 bytes): A 24-byte Unicode char field containing the short (8.3) file name. The file name might be NULL-terminated.

FileName (variable): A sequence of Unicode characters containing the file name. This field might not be NULL-terminated, and MUST be handled as a sequence of **FileNameLength** bytes.

2.4.9 FileCompressionInformation

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

The FILE_COMPRESSION_INFORMATION data element is as follows:



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 associate any local compression

algorithm with the values described below because the compressed data does not travel across the wire in the context of FSCTL, FileInformation class, or FileSystemInformation class requests or replies.<33>

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.
COMPRESSION_FORMAT_LZNT1 0x0002	The file or directory is compressed by using the LZNT1 compression algorithm.
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.<34>

ChunkShift (1 byte): An 8-bit unsigned integer that contains the compression **chunk** size in bytes in log 2 format. 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.<35>

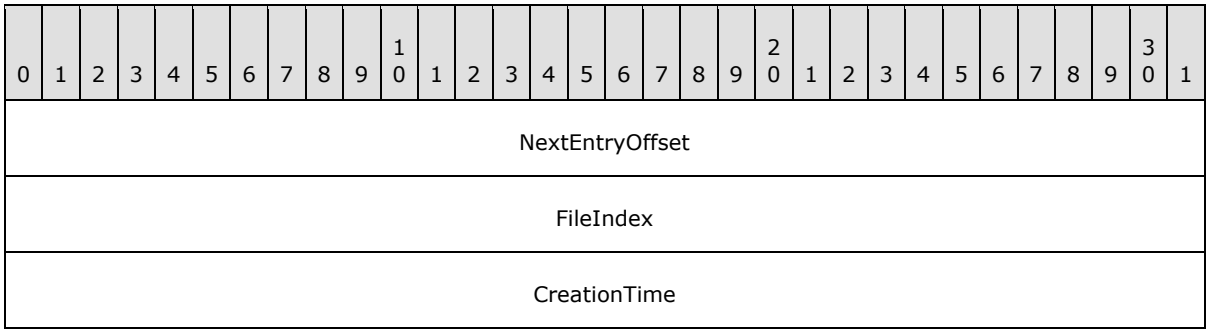
ClusterShift (1 byte): An 8-bit unsigned integer that specifies, in log 2 format, the amount of space that must be saved by compression to successfully compress a compression unit. If that amount of space is not saved by compression, the data in that compression unit is stored uncompressed. Each successfully compressed compression unit MUST occupy at least one cluster that is less in bytes than an uncompressed compression unit. Therefore, the cluster shift is the number of bits by which to left shift a 1 bit to arrive at the size of a cluster. This value is implementation defined.<36>

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

2.4.10 FileDirectoryInformation

This information class is used to query detailed information for the files in a directory.

The FILE_DIRECTORY_INFORMATION data element is as follows:



...
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), and MUST NOT assume that the value of NextEntryOffset is the same as the size of the current entry.

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.<37>

CreationTime (8 bytes): A 64-bit signed integer that contains the time when the file was created. All dates and times are in absolute system-time format, which is represented as a **FILETIME (section 2.1.1)** structure. This value MUST be greater than or equal to 0.

LastAccessTime (8 bytes): A 64-bit signed integer that contains the last time the file was accessed in the format of a **FILETIME** structure. This value MUST be greater than or equal to 0.

LastWriteTime (8 bytes): A 64-bit signed integer that contains the last time information was written to the file in the format of a **FILETIME** structure. This value MUST be greater than or equal to 0.

ChangeTime (8 bytes): A 64-bit signed integer that contains the last time the file was changed in the format of a **FILETIME** structure. 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. Usually, this value is a multiple of the sector or cluster size of the underlying physical device. 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.

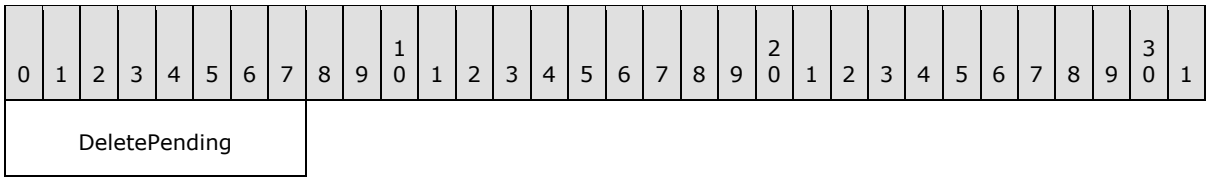
FileNameLength (4 bytes): A 32-bit unsigned integer that contains the length, in bytes, of the **FileName** field.

FileName (variable): A sequence of Unicode characters containing the file name. This field might not be NULL-terminated, and MUST be handled as a sequence of **FileNameLength** bytes.

2.4.11 FileDispositionInformation

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

The FILE_DISPOSITION_INFORMATION data element is as follows:

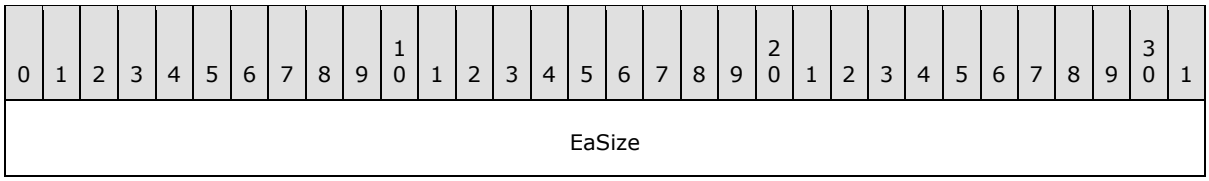


DeletePending (1 byte): An 8-bit field that is set to 1 to indicate that a file SHOULD be deleted when it is closed; otherwise, 0.<38>

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 associate with a file that is not part of the file's data. For more information about extended attributes, see [CIFS].

The FILE_EA_INFORMATION data element is as follows:

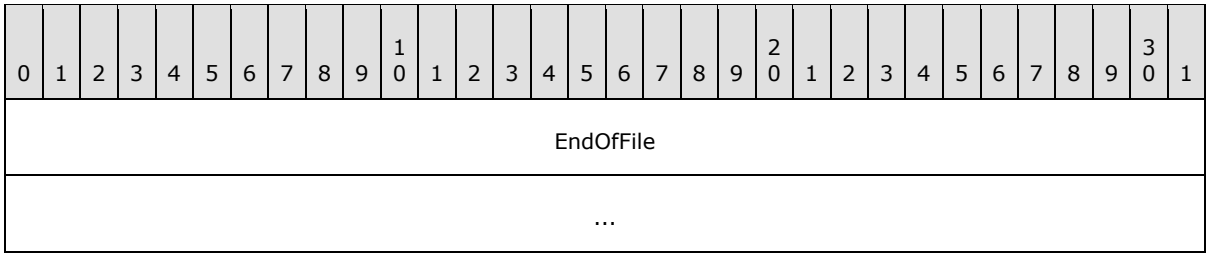


EaSize (4 bytes): A 32-bit unsigned integer that contains the combined length in bytes of the extended attributes (EA) for the file.

2.4.13 FileEndOfFileInformation

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

The FILE_END_OF_FILE_INFORMATION data element is as follows:

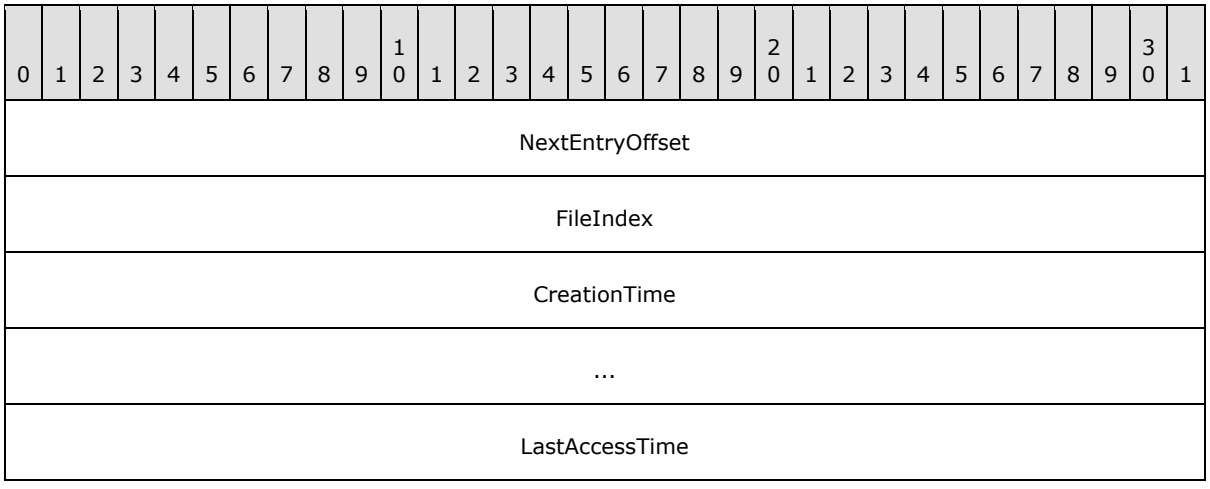


EndOfFile (8 bytes): A 64-bit signed integer that contains the absolute new end of file position as a byte offset from the start of the file. EndOfFile specifies the offset from the beginning of the file of the byte following the last byte in the file. That is, it is the offset from the beginning of 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.

2.4.14 FileFullDirectoryInformation

This information class is used to query detailed information for the files in a directory.

The FILE_FULL_DIR_INFORMATION data element is as follows:



...
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), and MUST NOT assume that the value of NextEntryOffset is the same as the size of the current entry.

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.<39>

CreationTime (8 bytes): A 64-bit signed integer that contains the time when the file was created in the format of a **FILETIME** structure. This value MUST be greater than or equal to 0.

LastAccessTime (8 bytes): A 64-bit signed integer that contains the last time the file was accessed in the format of a **FILETIME** structure. This value MUST be greater than or equal to 0.

LastWriteTime (8 bytes): A 64-bit signed integer that contains the last time information was written to the file in the format of a **FILETIME** structure. This value MUST be greater than or equal to 0.

ChangeTime (8 bytes): A 64-bit signed integer that contains the last time the file was changed in the format of a **FILETIME** structure. 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. Usually, this value is a multiple of the sector or cluster size of the underlying physical device. 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. For a list of valid file attributes, see section 2.6.

FileNameLength (4 bytes): A 32-bit unsigned integer that contains the length, in bytes, of the **FileName** field.

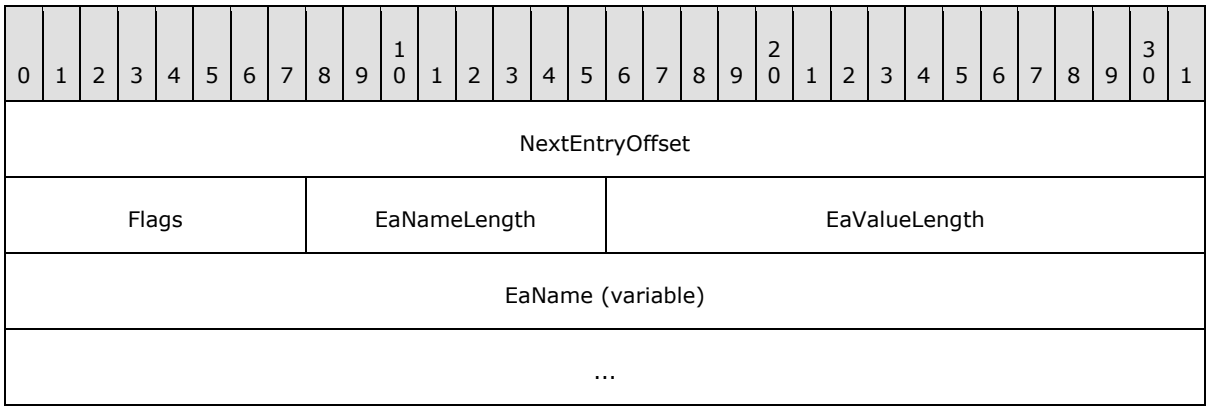
EaSize (4 bytes): A 32-bit unsigned integer that contains the combined length, in bytes, of the extended attributes (EA) for the file.

FileName (variable): A sequence of Unicode characters containing the file name. This field might not be NULL-terminated, and MUST be handled as a sequence of **FileNameLength** bytes.

2.4.15 FileFullEaInformation

This information class is used to query or set extended attribute (EA) information for a file.

The 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), and MUST NOT assume that the value of NextEntryOffset is the same as the size of the current entry.

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

Value	Meaning
0x00000000	The file does not use EAs.
FILE_NEED_EA 0x00000080	If this flag is set, the file to which the EA belongs cannot be interpreted without understanding the associated extended attributes.

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 does not include the null-terminator to **EaName**.

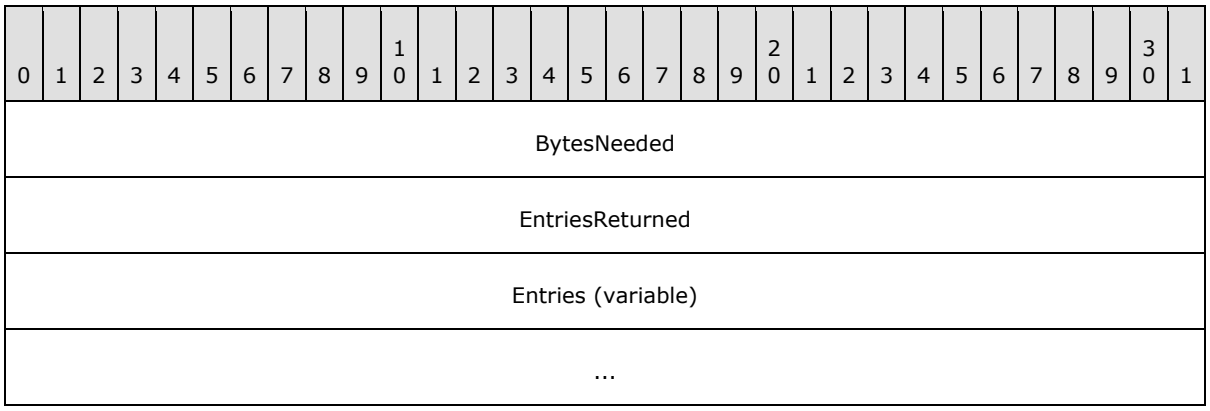
EaValueLength (2 bytes): A 16-bit unsigned integer that contains the length, in bytes, of the extended attribute value in the **EaName** field.

EaName (variable): An array of 8-bit ASCII characters that contain the extended attribute name followed by a single null-termination character byte and the associated value.

2.4.16 FileHardLinkInformation

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

The FILE_LINKS_INFORMATION data element is as follows:



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.

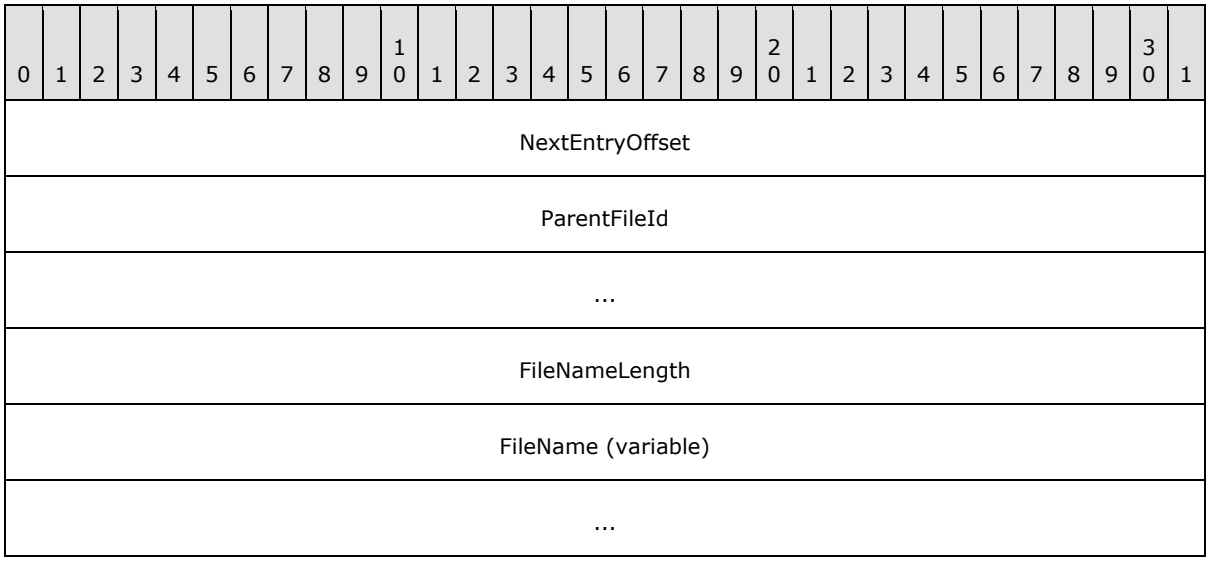
EntriesReturned (4 bytes): A 32-bit unsigned integer that MUST contain the number of FILE_LINK_ENTRY_INFORMATION structures that have been returned in the Entries field.

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 in size to return all of the available entries.

2.4.16.1 FILE_LINK_ENTRY_INFORMATION

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



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.

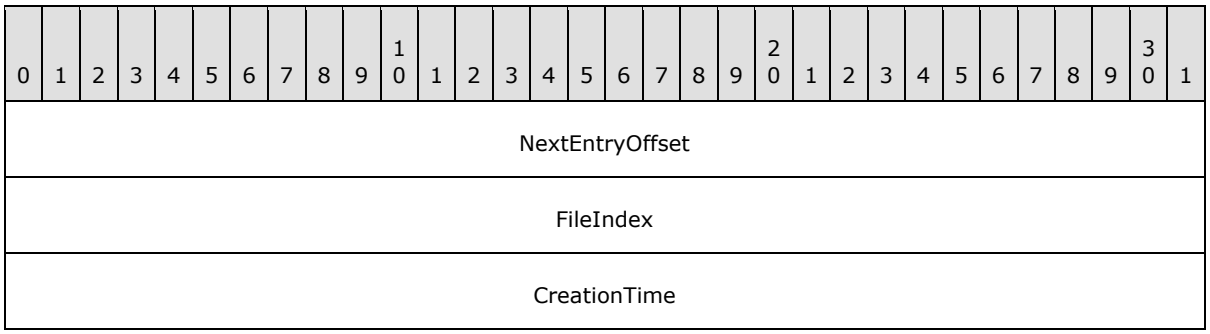
FileNameLength (4 bytes): A 32-bit unsigned integer that MUST specify the length, in bytes, of the FileName for the given link.

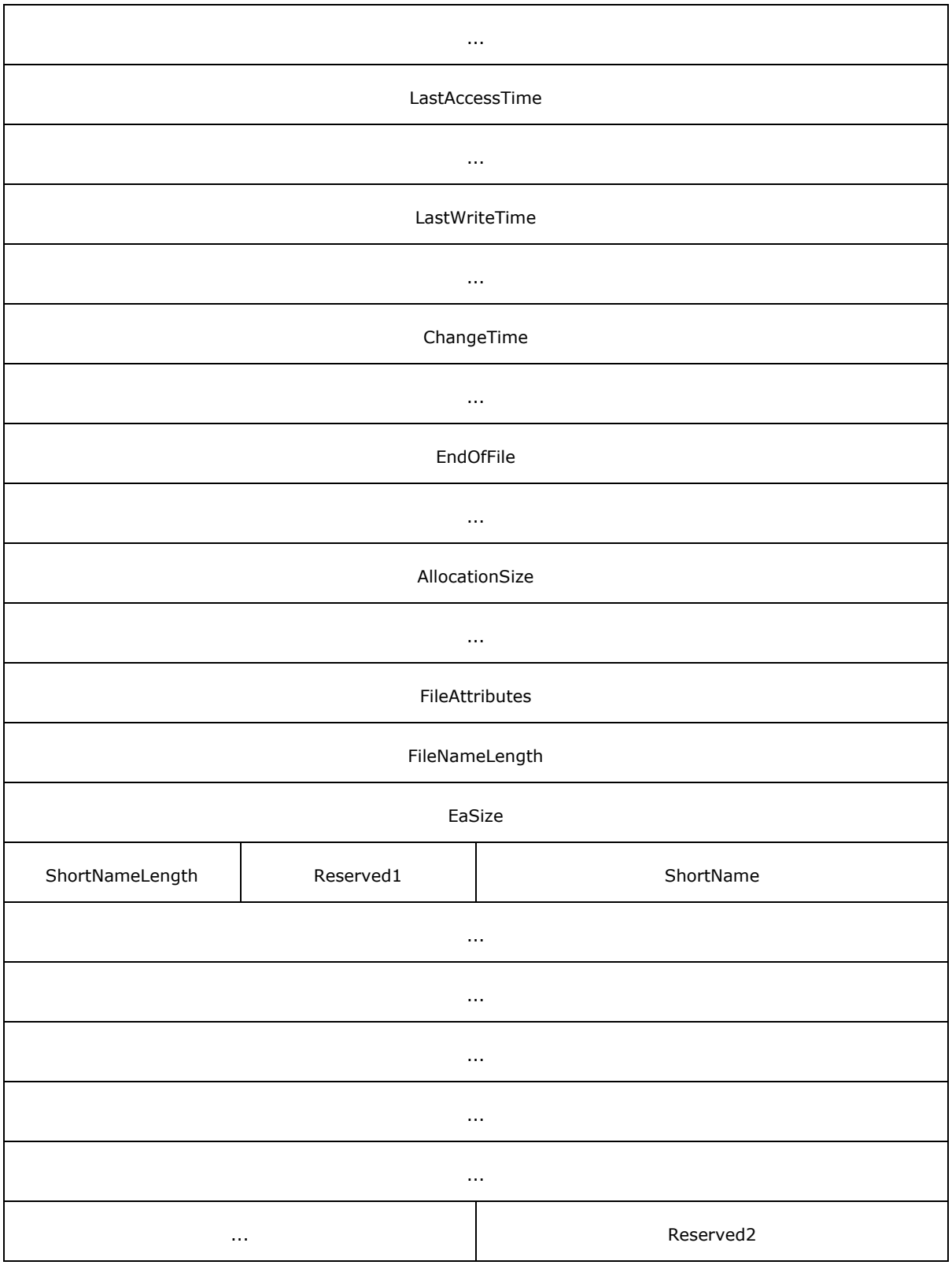
FileName (variable): A **WCHAR** array whose size is given by FileNameLength that MUST contain the Unicode string name of the given link.

2.4.17 FileIdBothDirectoryInformation

This information class is used to query file reference number information for the files in a directory.

The FILE_ID_BOTH_DIR_INFORMATION data element is as follows:





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), and MUST NOT assume that the value of NextEntryOffset is the same as the size of the current entry.

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.<40>

CreationTime (8 bytes): A 64-bit signed integer that contains the time when the file was created. All dates and times are in absolute system-time format, which is represented as a **FILETIME** structure. The value of this field MUST be greater than or equal to 0.

LastAccessTime (8 bytes): A 64-bit signed integer that contains the last time the file was accessed in the format of a **FILETIME** structure. The value of this field MUST be greater than or equal to 0.

LastWriteTime (8 bytes): A 64-bit signed integer that contains the last time information was written to the file in the format of a **FILETIME** structure. The value of this field MUST be greater than or equal to 0.

ChangeTime (8 bytes): A 64-bit signed integer that contains the last time the file was changed in the format of a **FILETIME** structure. 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. Usually, this value is a multiple of the sector or cluster size of the underlying physical device. 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.

FileNameLength (4 bytes): A 32-bit unsigned integer that contains the length, in bytes, of the **FileName** field.

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 value containing the length, in bytes, of the **ShortName** string.

Reserved1 (1 byte): MUST be ignored by the receiver.

ShortName (24 bytes): A NULL-terminated 12-character Unicode string containing the short (8.3) file name.

Reserved2 (2 bytes): MUST be ignored by the receiver.

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. Note that the FileId is not the same as the 16-byte file object ID that was added to NTFS for Windows 2000.

FileName (variable): A sequence of Unicode characters containing the file name. This field might not be NULL-terminated, and MUST be handled as a sequence of **FileNameLength** bytes.

2.4.18 FileIdFullDirectoryInformation

This information class is used to query detailed information for the files in a directory.

The FILE_ID_FULL_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
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 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), and MUST NOT assume that the value of NextEntryOffset is the same as the size of the current entry.

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.<41>

CreationTime (8 bytes): A 64-bit signed integer that contains the time when the file was created in the format of a **FILETIME** structure. The value of this field MUST be greater than or equal to 0.

LastAccessTime (8 bytes): A 64-bit signed integer that contains the last time the file was accessed in the format of a **FILETIME** structure. The value of this field MUST be greater than or equal to 0.

LastWriteTime (8 bytes): A 64-bit signed integer that contains the last time information was written to the file in the format of a **FILETIME** structure. The value of this field MUST be greater than or equal to 0.

ChangeTime (8 bytes): A 64-bit signed integer that contains the last time the file was changed in the format of a **FILETIME** structure. 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. Usually, this value is a multiple of the sector or cluster size of the underlying physical device. 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.

FileNameLength (4 bytes): A 32-bit unsigned integer that contains the length, in bytes, of the **FileName** field.

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.

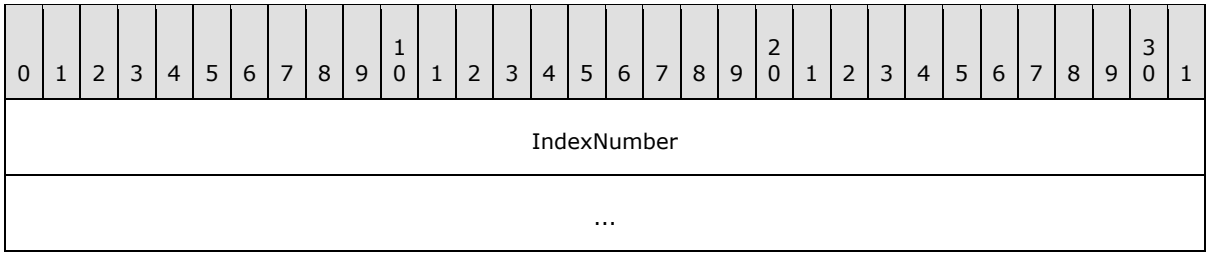
FileId (8 bytes): An 8-byte file reference number for the file. This number is generated and assigned to the file by the file system. Note that the FileId is not the same as the 16-byte file object ID that was added to NTFS for Windows 2000.

FileName (variable): A sequence of Unicode characters containing the file name. This field might not be NULL-terminated, and MUST be handled as a sequence of **FileNameLength** bytes.

2.4.19 FileInternalInformation

This information class is used to query for the file system's 8-byte file reference number for a file.

The FILE_INTERNAL_INFORMATION data element is as follows:

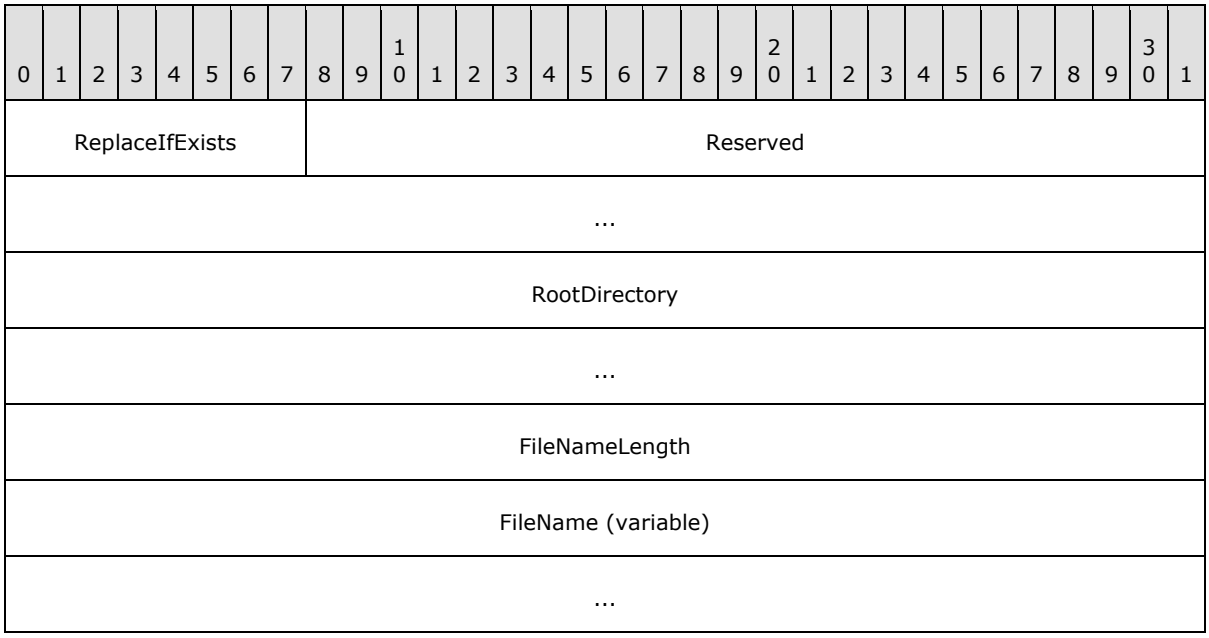


IndexNumber (8 bytes): A 64-bit signed integer that contains the 8-byte file reference number for the file. This number MUST be assigned by the file system and is unique to the volume on which the file or directory is located. This file reference number is the same as the file reference number that is 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.

2.4.20 FileLinkInformation

This information class is used to create an NTFS hard link to an existing file.

The FILE_LINK_INFORMATION data element is as follows:



ReplaceIfExists (1 byte): An 8-bit field that is set to 1 to indicate that if the link already exists, it should be replaced with the new link. MUST be set to 0 if the caller wants the link creation operation to fail if the link already exists.

Reserved (7 bytes): Reserved for alignment.

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.

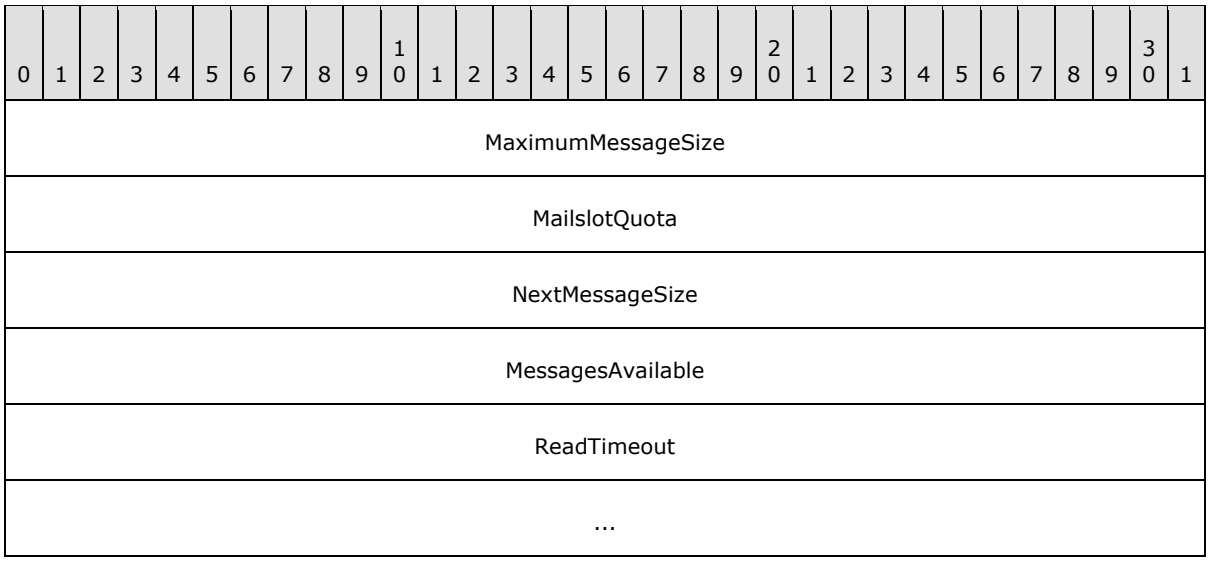
FileNameLength (4 bytes): A 32-bit unsigned integer that contains the length, in bytes, of the **FileName** field.

FileName (variable): A sequence of Unicode characters containing the name to be assigned to the newly created link. This field might not be NULL-terminated, and MUST be handled as a sequence of **FileNameLength** bytes. If the **RootDirectory** member is NULL, and the link is to be created in a different directory from the file that is being linked to, this member specifies the full path name for the link to be created. Otherwise, it specifies only the file name.

2.4.21 FileMailslotQueryInformation

This information class is used to query information on a mailslot.

The FILE_MAILSLOT_QUERY_INFORMATION data element is as follows:



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.

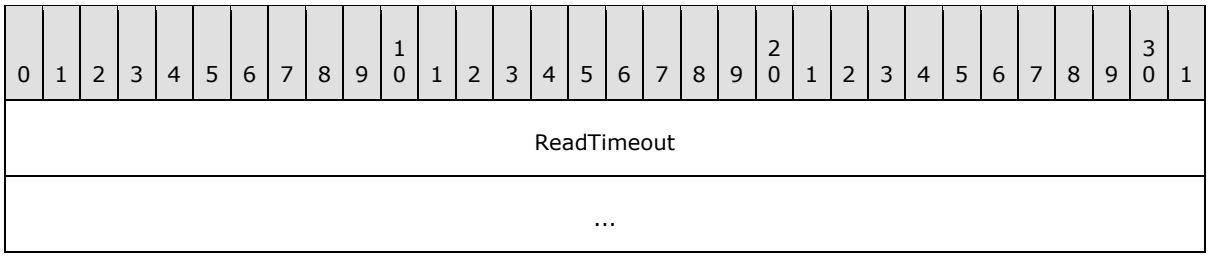
MessagesAvailable (4 bytes): A 32-bit unsigned integer that contains the total number of messages waiting to be read from the mailslot.

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.

2.4.22 FileMailslotSetInformation

This information class is used to set information on a mailslot.

The FILE_MAILSLOT_SET_INFORMATION data element is as follows:

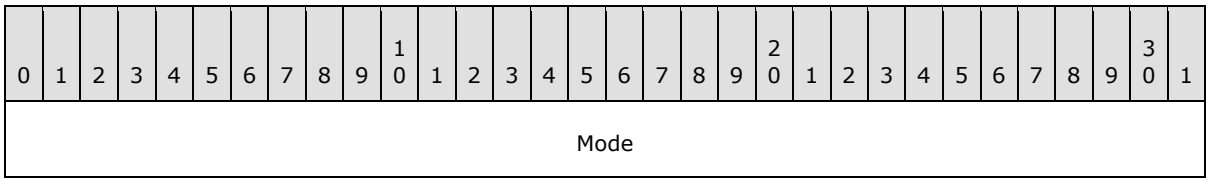


ReadTimeout (8 bytes): A 64-bit signed integer that contains the time a read operation can wait for a message to be written to the mailslot before a time-out occurs in milliseconds. The value of this field MUST be (-1) or greater than or equal to 0. A value of (-1) requests that the read wait forever 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.

2.4.23 FileModeInformation

This information class is used to query or set the mode of the file.

The FILE_MODE_INFORMATION data element is as follows:



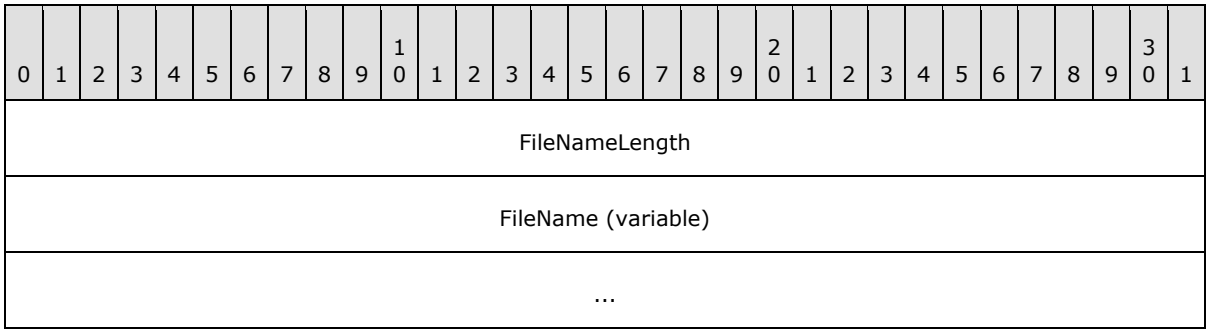
Mode (4 bytes): A **ULONG** that MUST specify on file creation or file open how the file will subsequently be accessed.

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	When set, all accesses to the file will be sequential.
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	When set, delete the file when the last handle to the file is closed.

2.4.24 FileNameInformation

This information class is used to query detailed information of a file.

The FILE_NAME_INFORMATION data element is as follows:



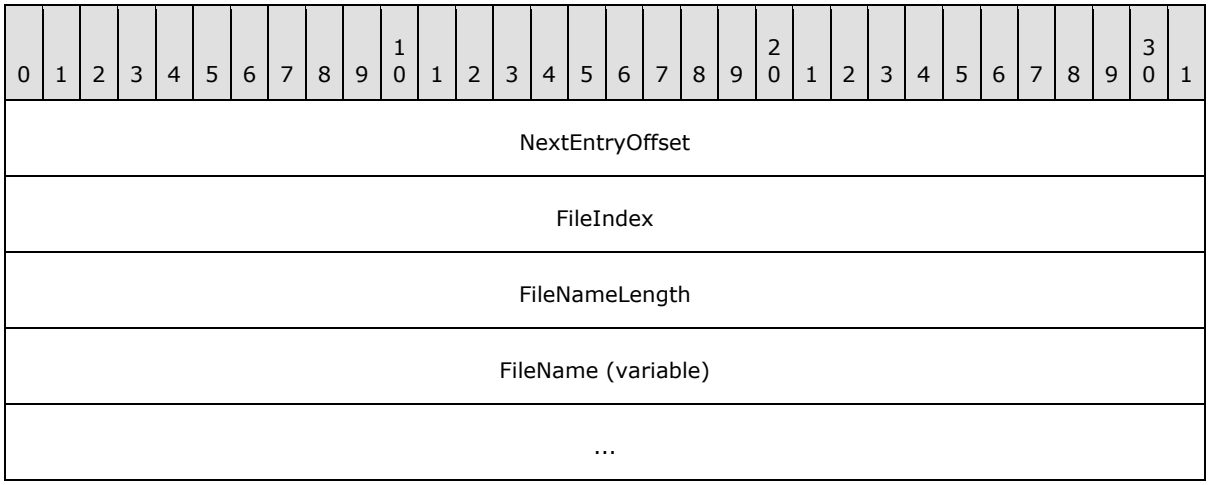
FileNameLength (4 bytes): A 32-bit unsigned integer that contains the length, in bytes, of the **FileName** field.

FileName (variable): A sequence of Unicode characters containing the file name. This field might not be NULL-terminated, and MUST be handled as a sequence of **FileNameLength** bytes.

2.4.25 FileNamesInformation

This information class is used to query detailed information on the names of files in a directory.

The 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), and MUST NOT assume that the value of NextEntryOffset is the same as the size of the current entry.

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.<42>

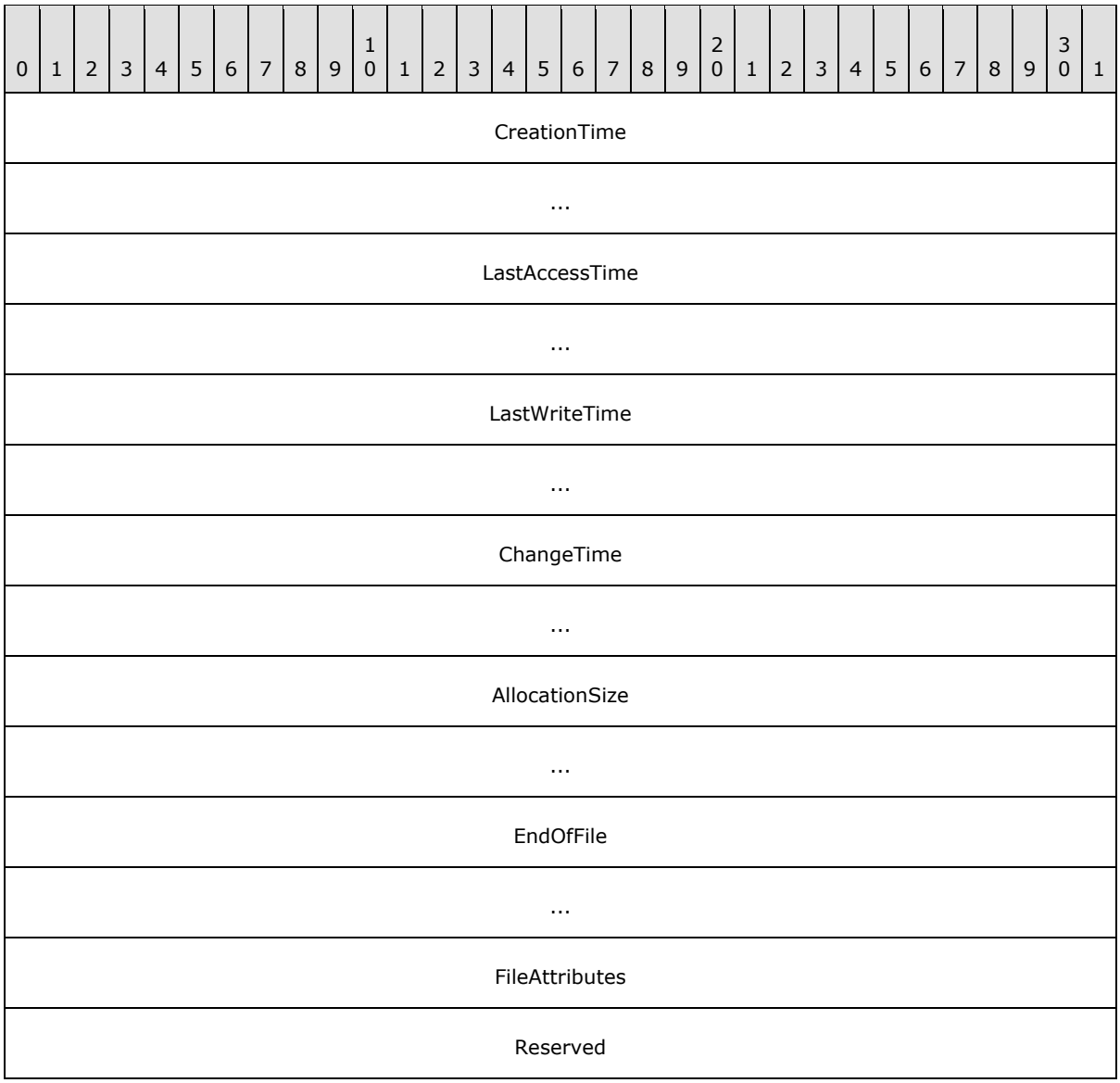
FileNameLength (4 bytes): A 32-bit unsigned integer that contains the length, in bytes, of the **FileName** field.

FileName (variable): A sequence of Unicode characters containing the file name. This field might not be NULL-terminated, and MUST be handled as a sequence of **FileNameLength** bytes.

2.4.26 FileNetworkOpenInformation

This information class is used to query for information on a network file open. A network file open differs from a file open in that the handle obtained from a network file open can be used to look up attributes using FileNetworkOpenInformation, but it cannot be used for reads and writes to the file. The network file open is an optimization of file open that returns a file handle to the caller more quickly, but the file handle it returns cannot be used in all of the ways that a normal file handle can be used.

The FILE_NETWORK_OPEN_INFORMATION data element is as follows:



CreationTime (8 bytes): A 64-bit signed integer that contains the time when the file was created in the format of a **FILETIME** structure. The value of this field MUST be greater than or equal to 0.

LastAccessTime (8 bytes): A 64-bit signed integer that contains the last time the file was accessed in the format of a **FILETIME** structure. The value of this field MUST be greater than or equal to 0.

LastWriteTime (8 bytes): A 64-bit signed integer that contains the last time information was written to the file in the format of a **FILETIME** structure. The value of this field MUST be greater than or equal to 0.

ChangeTime (8 bytes): A 64-bit signed integer that contains the last time the file was changed in the format of a **FILETIME** structure. 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. Usually, this value is a multiple of the sector or cluster size of the underlying physical device. 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.

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 MAY be set to any value, and MUST be ignored.

2.4.27 FileObjectIdInformation

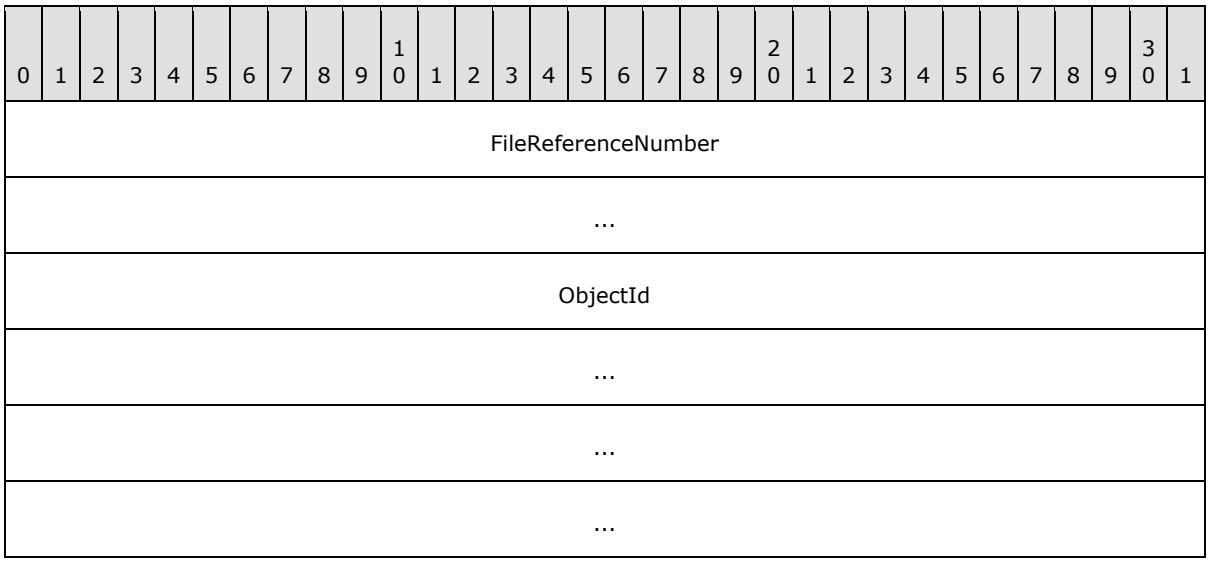
This information class is used to query object ID information for the files in a directory on a volume. The query MUST fail if the file system does not support object IDs.<43>

The FILE_OBJECTID_INFORMATION data element can contain two different types of data. 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.<44>

- FileObjectIdInformation Type 1
- FileObjectIdInformation Type 2

2.4.27.1 FileObjectIdInformation Type 1

The first type of data that may be returned.



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.<45>

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

2.4.27.2 FileObjectIdInformation Type 2

The second type of data that may be returned.

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
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

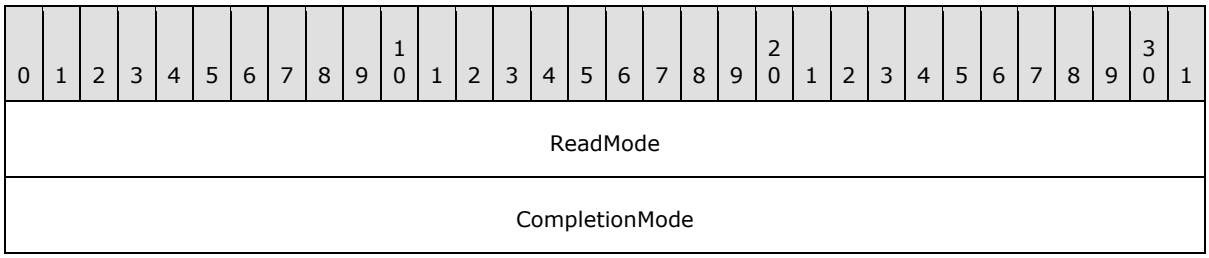
directory on a different volume, but it MUST NOT be assigned to another file or directory on the same volume.

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.

2.4.28 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.

The FILE_PIPE_INFORMATION data element is as follows:



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.

Note If the ReadMode of a pipe is set to FILE_PIPE_BYTE_STREAM_MODE, it cannot be subsequently changed.

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 mean waiting indefinitely in some situations 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 written from, the operation completes immediately.

2.4.29 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.

The FILE_PIPE_LOCAL_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
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.

Value	Meaning
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. The first instance of the pipe MUST specify this value.

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.

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 outbound quota in bytes for the named pipe.

WriteQuotaAvailable (4 bytes): A 32-bit unsigned integer that contains the write quota in bytes for the named pipe.

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.

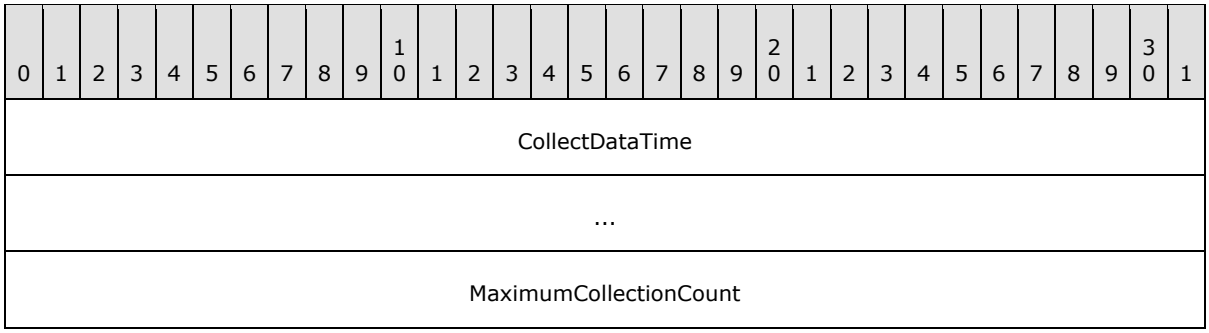
NamedPipeEnd (4 bytes): A 32-bit unsigned integer that contains the type of the named pipe end, which specifies whether this is the client or the server side of a named pipe. MUST be one of the following.

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.

2.4.30 FilePipeRemoteInformation

This information class is used to query or set 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.

The FILE_PIPE_REMOTE_INFORMATION data element is as follows:



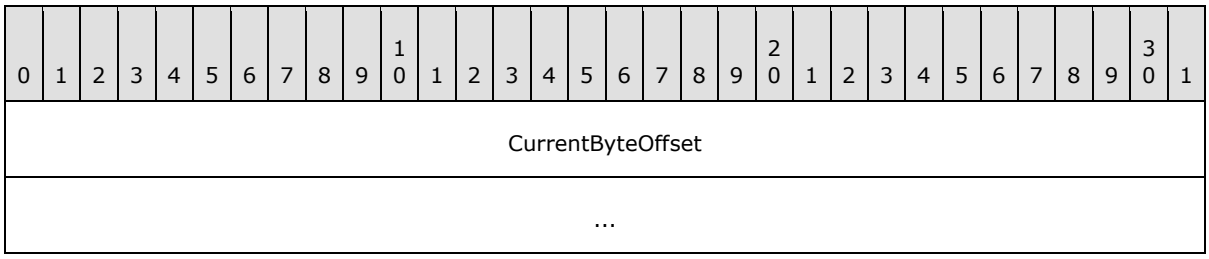
CollectDataTime (8 bytes): A **LARGE_INTEGER** that MUST contain the maximum amount of time counted in 100-nanosecond intervals that will elapse before transmission of data from the client machine to the server.

MaximumCollectionCount (4 bytes): A **ULONG** that MUST contain the maximum size in bytes of data that will be collected on the client machine before transmission to the server.

2.4.31 FilePositionInformation

This information class is used to query the position of the file pointer within a file.

The FILE_POSITION_INFORMATION data element is as follows:

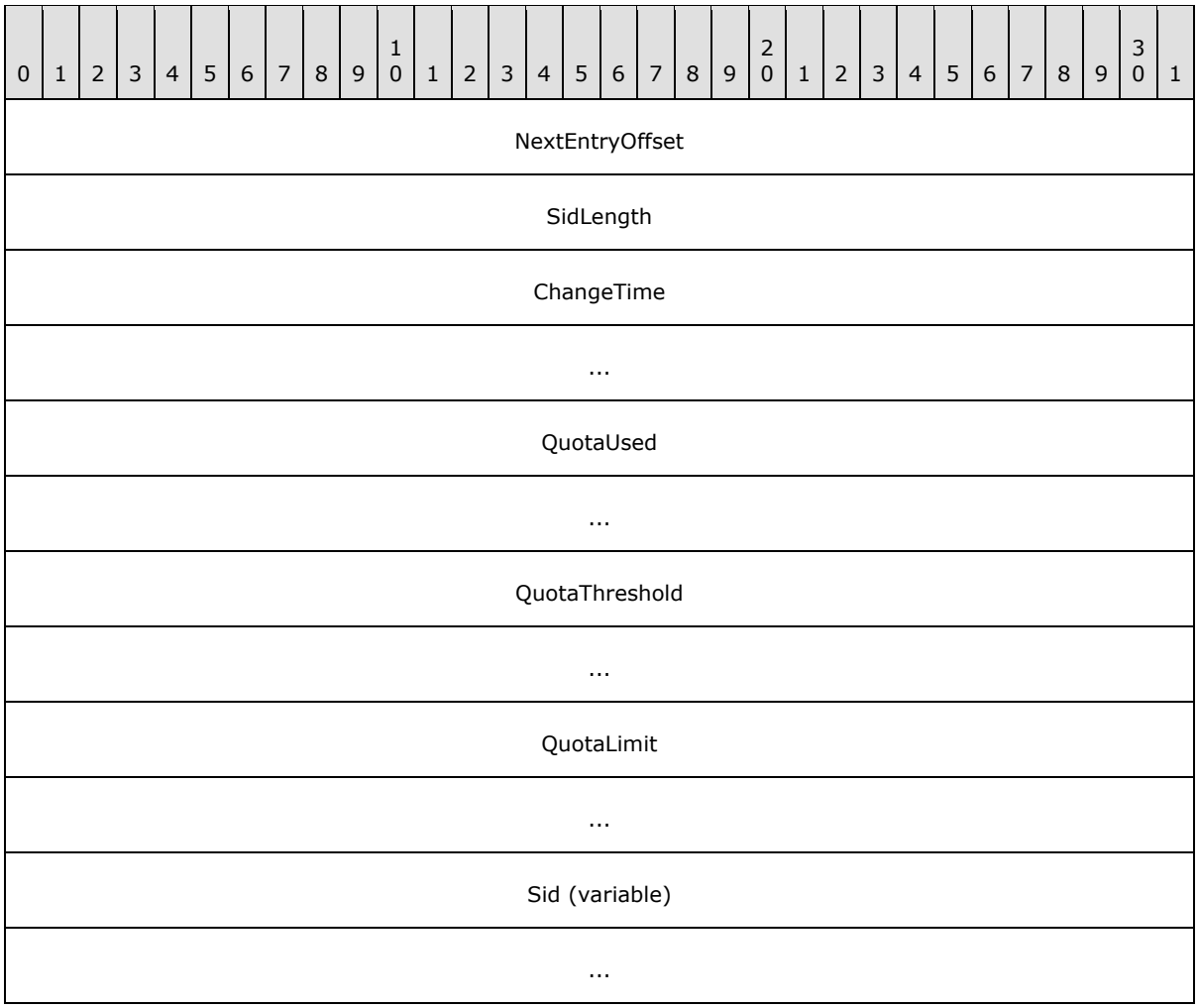


CurrentByteOffset (8 bytes): A **LARGE_INTEGER** that MUST contain the offset, in bytes, of the file pointer from the beginning of the file.

2.4.32 FileQuotaInformation

The information class is used to query or set quota information.

The FILE_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_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), and MUST NOT assume that the value of NextEntryOffset is the same as the size of the current entry.

SidLength (4 bytes): A 32-bit unsigned integer that contains the length, in bytes, of the SID data element.

ChangeTime (8 bytes): A 64-bit signed integer that contains the last time the quota was changed in the format of a **FILETIME** structure. This value MUST be greater than or equal to 0.

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 0.

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

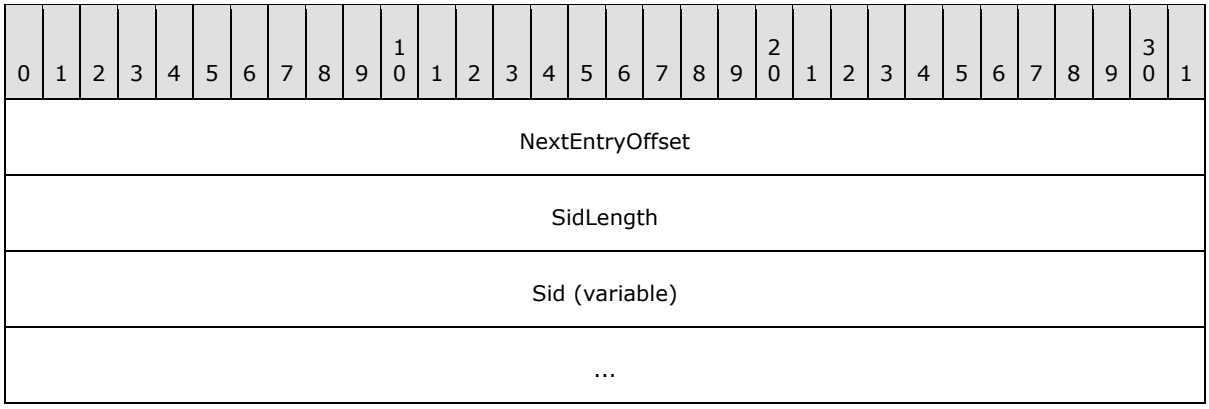
greater than or equal to 0 to set a quota warning threshold for this user on this volume, or to (-1) to specify that no quota warning threshold is set for this user.

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 0 to set a disk quota limit for this user on this volume, or to (-1) to specify that no quota limit is set for this user.

Sid (variable): Security identifier (SID) for this user.

2.4.32.1 FileGetQuotaInformation

The information class is used to provide the list of Security identifiers (SID) for which query quota information is requested.



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), and MUST NOT assume that the value of NextEntryOffset is the same as the size of the current entry.

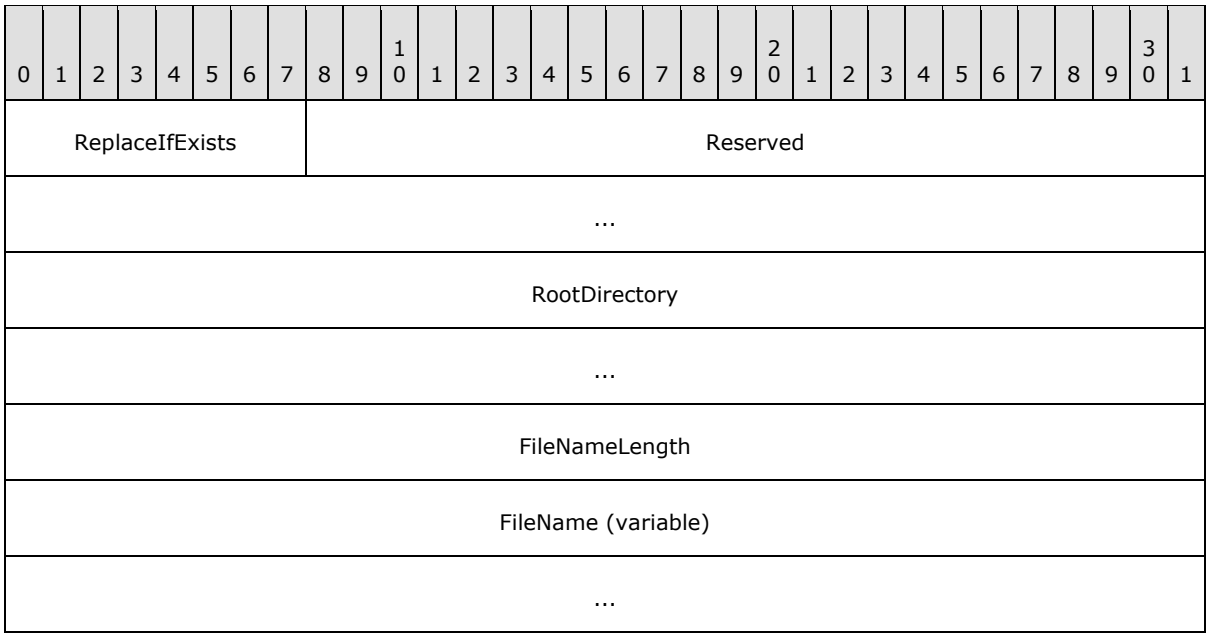
SidLength (4 bytes): A 32-bit unsigned integer that contains the length, in bytes, of the <Sid> data element.

Sid (variable): SIDfor this user. SID(s) are sent in little endian format and require no packing. The format of a **SID** is as specified in **[MS-DTYP]** section **2.4.2**.

2.4.33 FileRenameInformation

This information class is used to rename a file.

The FILE_RENAME_INFORMATION data element is as follows:



ReplaceIfExists (1 byte): MUST be an 8-bit field that is set to 1 to indicate that if a file with the given name already exists, it SHOULD be replaced with the given file. If set to 0, the rename operation MUST fail if a file with the given name already exists.

Reserved (7 bytes): Reserved area for alignment.

RootDirectory (8 bytes): A 64-bit unsigned integer that contains the file handle for the root directory. 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 new name for the file, including the trailing NULL if present.

FileName (variable): A sequence of Unicode characters containing the file name. This field MAY NOT be NULL-terminated, and MUST be handled as a sequence of **FileNameLength** bytes, not as a NULL-terminated string. If the **RootDirectory** member is NULL, and the file is being moved to a different directory, this member MUST specify the full path name to be assigned to the file. Otherwise, it MUST specify only the file name or a relative path name.

2.4.34 FileReparsePointInformation

This information class is used to query for information on a reparse point.

The FILE_REPARSE_POINT_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
FileReferenceNumber																															
...																															
Tag																															

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.

2.4.35 FileShortNameInformation

This information class is used to query or change the file's short name. A caller changing the file's short name MUST have SeBackupPrivilege. SeBackupPrivilege is specified in the Windows Behavior note of [MS-LSAD] section 3.1.1.2.1.

The FILE_NAME_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
FileNameLength																															
FileName (variable)																															
...																															

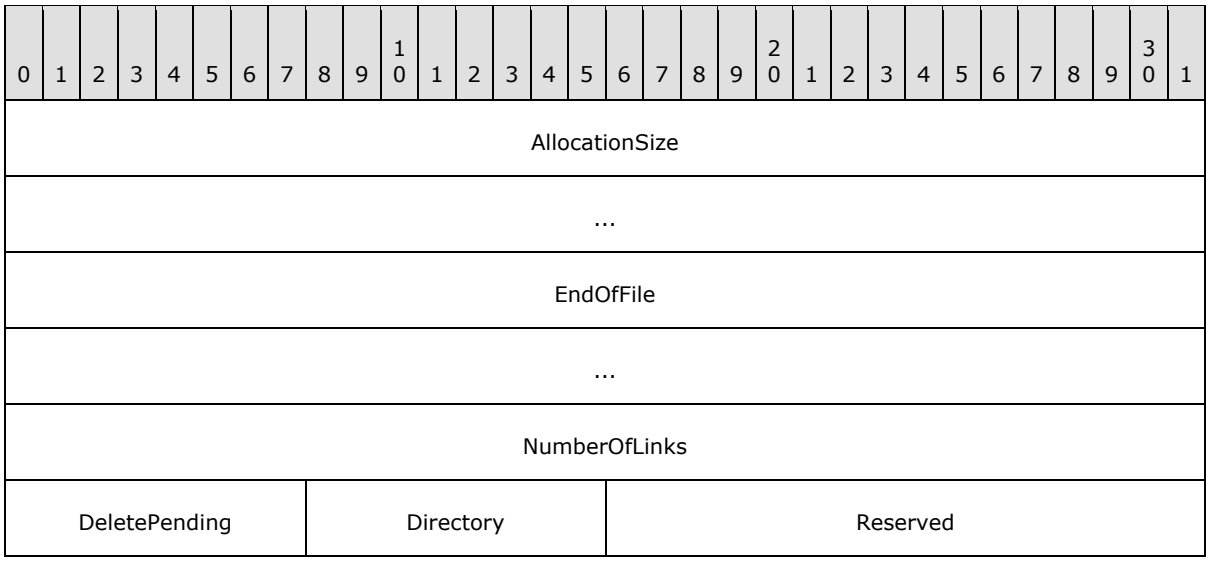
FileNameLength (4 bytes): A 32-bit unsigned integer that contains the length, in bytes, of the **FileName** field.

FileName (variable): A sequence of Unicode characters containing the file name. This field MUST NOT begin with a path separator character (backslash). This field might not be NULL-terminated, and MUST be handled as a sequence of **FileNameLength** bytes.

2.4.36 FileStandardInformation

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

The FILE_STANDARD_INFORMATION data element is as follows:



AllocationSize (8 bytes): A 64-bit signed integer that contains the file allocation size in bytes. Usually, this value is a multiple of the sector or cluster size of the underlying physical device. 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.

NumberOfLinks (4 bytes): A 32-bit unsigned integer that contains the number of non-deleted links to this file.

DeletePending (1 byte): An 8-bit field that MUST be set to 1 to indicate that a file deletion has been requested; otherwise, 0.

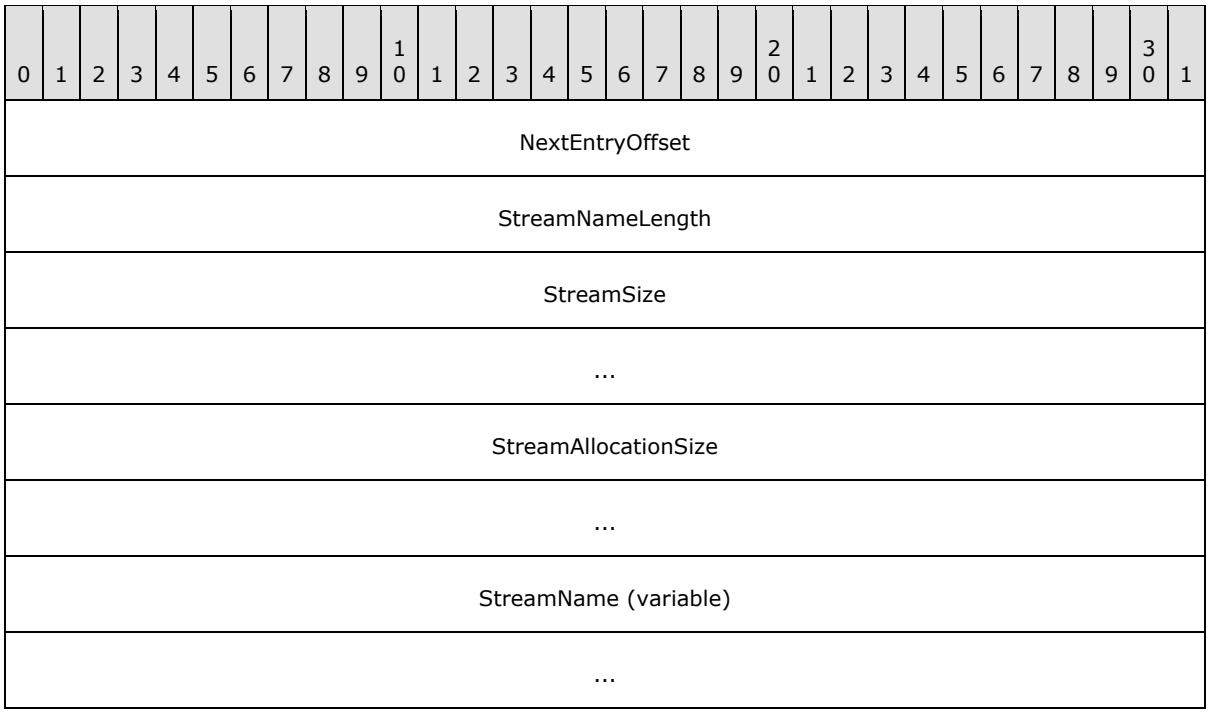
Directory (1 byte): An 8-bit field that MUST be set to 1 to indicate that the file is a directory; otherwise, 0.

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

2.4.37 FileStreamInformation

This information class is used to enumerate the streams for a file.

The FILE_STREAM_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_ 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), and MUST NOT assume that the value of NextEntryOffset is the same as the size of the current entry.

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 unsigned integer that contains the size, in bytes, of the stream.

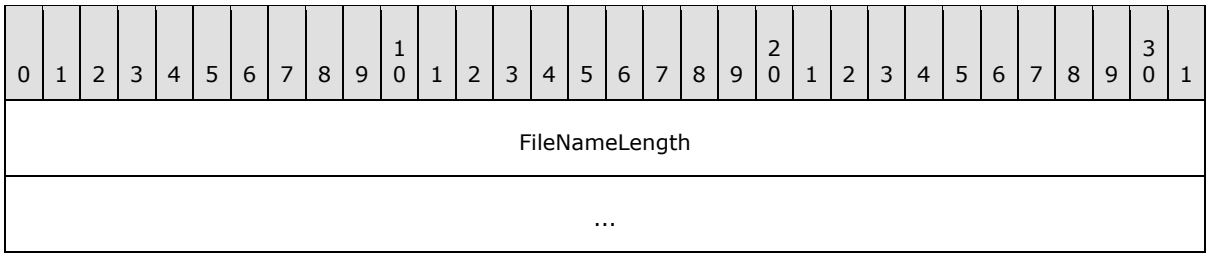
StreamAllocationSize (8 bytes): A 64-bit signed integer that contains the file stream allocation size in bytes. Usually, this value is a multiple of the sector or cluster size of the underlying physical device. The value of this field MUST be greater than or equal to 0.

StreamName (variable): A sequence of Unicode characters containing the name of the stream. This field might not be NULL-terminated, and MUST be handled as a sequence of **StreamNameLength** bytes.

2.4.38 FileValidDataLengthInformation

This information class is used to set the valid data length information for a file.

The FILE_VALID_DATA_LENGTH_INFORMATION data element is as follows:



FileNameLength (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.<46>

2.5 File System Information Classes

File system information classes are numerical values (specified by the Level column in the table below) 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.<47>

File system information class	Level
FileFsVolumeInformation	1
FileFsLabelInformation	2
FileFsSizeInformation	3
FileFsDeviceInformation	4
FileFsAttributeInformation	5
FileFsControlInformation	6
FileFsFullSizeInformation	7
FileFsObjectIdInformation	8
FileFsDriverPathInformation	9

2.5.1 FileFsAttributeInformation

This information class is used to query attribute information for a file system. The message contains a FILE_FS_ATTRIBUTE_INFORMATION data element.

The FILE_FS_ATTRIBUTE_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
FileSystemAttributes																															
MaximumComponentNameLength																															
FileSystemNameLength																															
FileSystemName (variable)																															
...																															

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 FS_FILE_COMPRESSION and FS_VOL_IS_COMPRESSED cannot both be set. Any flag values not explicitly mentioned here MAY be set to any value, and MUST be ignored.

Value	Meaning
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.
FILE_FILE_COMPRESSION 0x00000010	The file volume supports file-based compression. This flag is incompatible with the FILE_VOLUME_IS_COMPRESSED flag.
FILE_NAMED_STREAMS 0x00040000	The file system supports named streams.
FILE_PERSISTENT_ACLS 0x00000008	The file system preserves and enforces access control lists (ACLs).
FILE_READ_ONLY_VOLUME 0x00080000	The specified volume is read-only.<48>
FILE_SUPPORTS_ENCRYPTION 0x00020000	The file system supports the Encrypted File System (EFS). NTFS version 2 supports EFS. To use EFS, the operating system must support NTFS version 2, and the file system on disk must be formatted using NTFS version 2.<49>
FILE_SUPPORTS_OBJECT_IDS 0x00010000	The file system supports object identifiers.
FILE_SUPPORTS_REMOTE_STORAGE 0x00000100	The file system supports remote storage.
FILE_SUPPORTS_REPARSE_POINTS 0x00000080	The file system supports reparse points.

Value	Meaning
FILE_SUPPORTS_SPARSE_FILES 0x00000040	The file system supports sparse files.
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_VOLUME_IS_COMPRESSED 0x00008000	The specified volume is a compressed volume. This flag is incompatible with the FILE_FILE_COMPRESSION flag.
FILE_VOLUME_QUOTAS 0x00000020	The file system supports per-user quotas.

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 0.

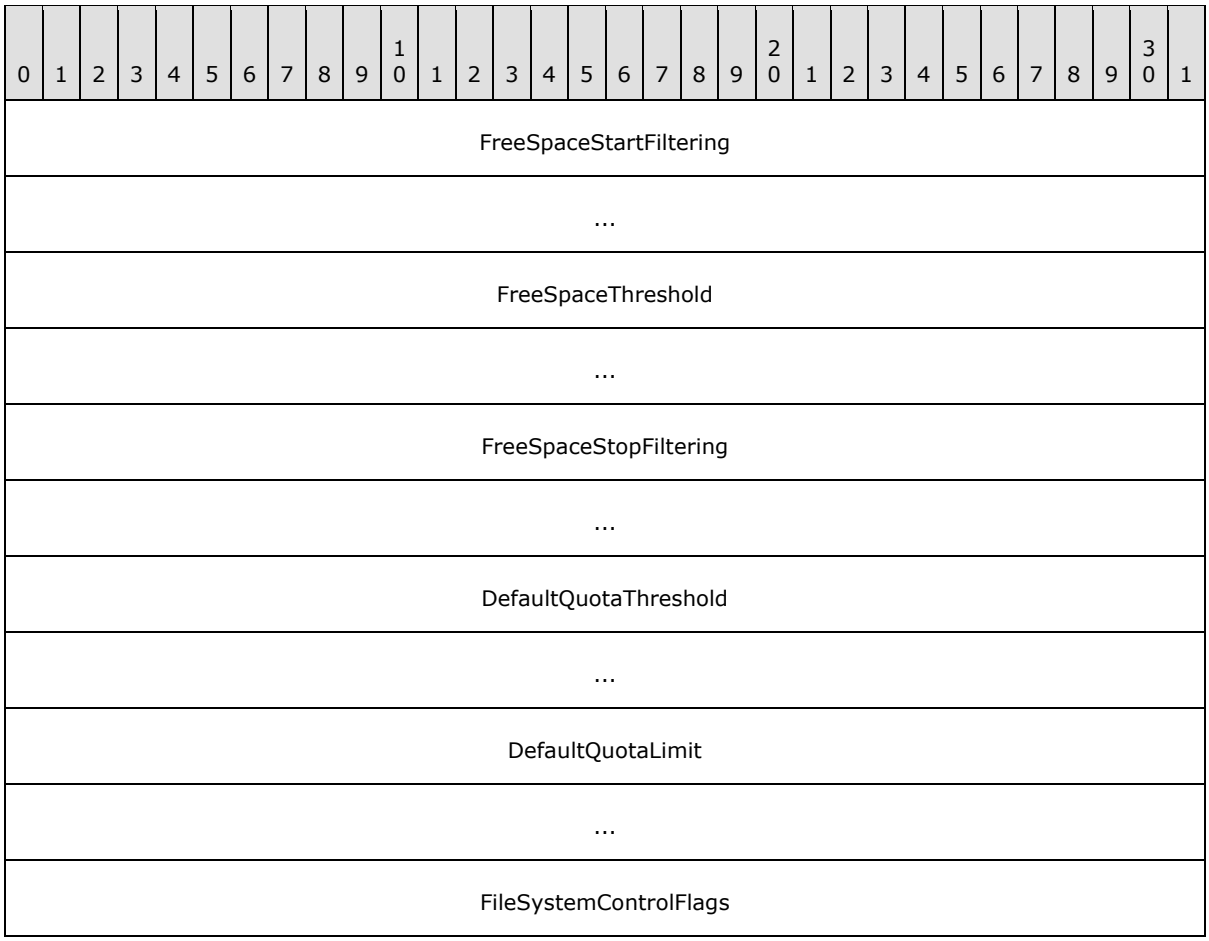
FileSystemNameLength (4 bytes): A 32-bit unsigned integer that contains the length, in bytes, of the file system name in the **FileSystemName** field. The value of this field MUST be greater than 0.

FileSystemName (variable): A variable-length Unicode field containing the name of the file system. This field might not be NULL-terminated, and MUST be handled as a sequence of **FileSystemNameLength** bytes.

2.5.2 FileFsControlInformation

This information class is used to query or set quota and content indexing control information for a file system volume. The message contains a FILE_FS_CONTROL_INFORMATION data element.<50>

The FILE_FS_CONTROL_INFORMATION data element is as follows:



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.<51>

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.<52>

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.<53>

DefaultQuotaThreshold (8 bytes): A 64-bit signed integer that contains the default per-user disk quota warning threshold in bytes for the volume. This field MUST be set to a 64-bit integer value greater than or equal to 0 to set a default quota warning threshold per user for this volume, or to (-1) to specify that no default quota warning threshold per user is set.

DefaultQuotaLimit (8 bytes): A 64-bit signed integer that contains the default per-user disk quota limit in bytes for the volume. This field MUST be set to a 64-bit integer value greater than or equal to 0 to set a default disk quota limit per user for this volume, or to (-1) to specify that no default quota limit per user is set.

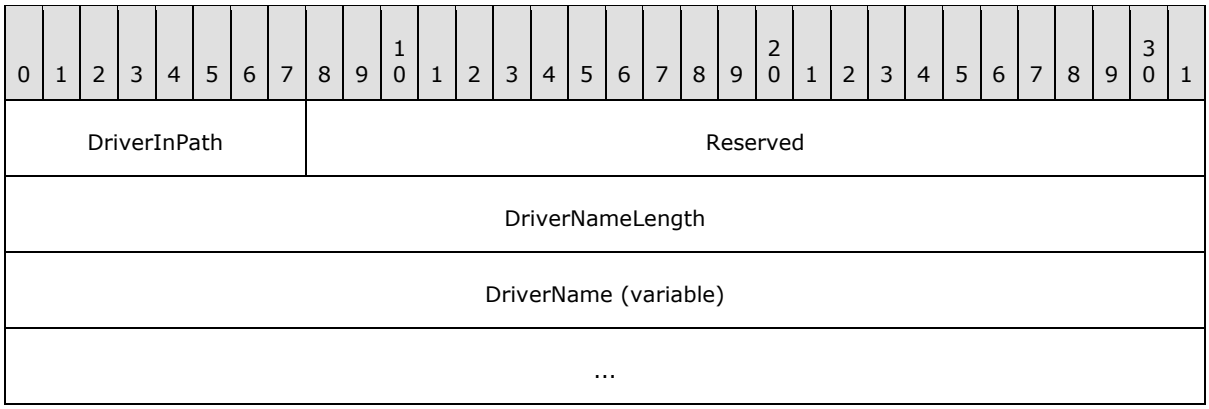
FileSystemControlFlags (4 bytes): A 32-bit unsigned integer that contains a bitmask of flags that control quota enforcement and logging of user-related quota events on the volume. The following bit flags are valid in any combination. Bits not defined below SHOULD be set to 0, and MUST be ignored.<54><55>

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 his or her 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.
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.
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 field and FILE_VC_QUOTA_ENFORCE are specified, FILE_VC_QUOTA_ENFORCE is ignored.
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.
FILE_VC_QUOTAS_REBUILDING 0x00000200	The file system is rebuilding the quota information for the volume.

2.5.3 FileFsDriverPathInformation

This information class is used to query if a given driver is in the I/O path for a file system volume. The message contains a FILE_FS_DRIVER_PATH_INFORMATION data element.

The FILE_FS_DRIVER_PATH_INFORMATION data element is as follows:



DriverInPath (1 byte): An unsigned character (Boolean) value that is TRUE if the driver is in the I/O path for the file system volume; and otherwise, FALSE.

Reserved (3 bytes): Reserved for alignment.

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.

2.5.4 FileFsFullSizeInformation

This information class is used to query sector size information for a file system volume. The message contains a FILE_FS_FULL_SIZE_INFORMATION data element.

The FILE_FS_FULL_SIZE_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
TotalAllocationUnits																															
...																															
CallerAvailableAllocationUnits																															
...																															
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.<56>

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.<57>

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.

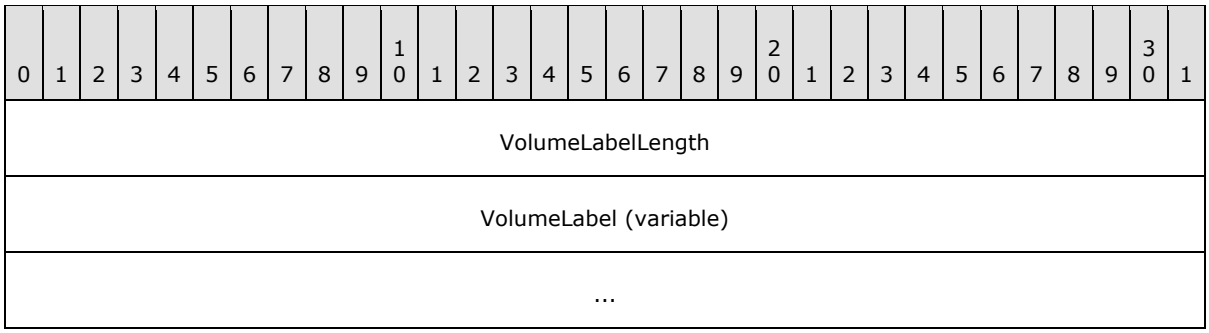
SectorsPerAllocationUnit (4 bytes): A 32-bit unsigned integer that contains the number of sectors in each allocation unit.

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

2.5.5 FileFsLabelInformation

This information class is used to set the label for a file system volume. The message contains a FILE_FS_LABEL_INFORMATION data element.

The FILE_FS_LABEL_INFORMATION data element is as follows:



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.

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.

2.5.6 FileFsObjectIdInformation

This information class is used to query or set the object ID for a file system volume. The message contains a FILE_FS_OBJECTID_INFORMATION data element.

The FILE_FS_OBJECTID_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
ObjectId																															
...																															
...																															
...																															
ExtendedInfo																															
...																															
...																															
...																															
...																															
...																															
...																															
...																															
(ExtendedInfo cont'd for 4 rows)																															

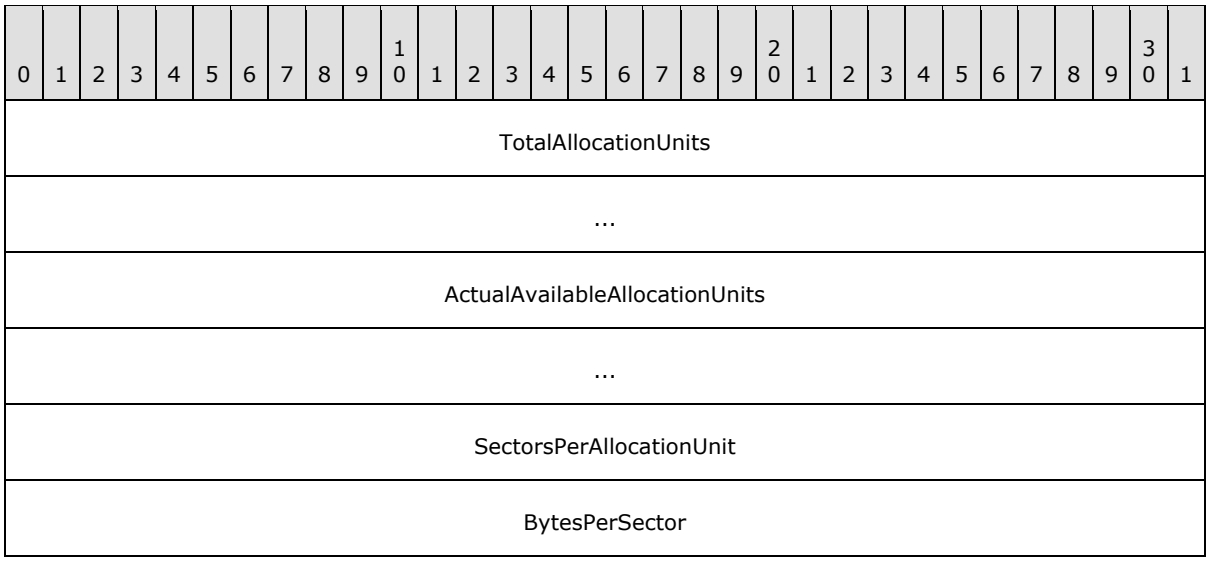
ObjectId (16 bytes): A 16-byte GUID that identifies the file system volume on the disk. This value is not required to be unique on the system.

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.<58>

2.5.7 FileFsSizeInformation

This information class is used to query sector size information for a file system volume. The message contains a FILE_FS_SIZE_INFORMATION data element.

The FILE_FS_SIZE_INFORMATION data element is as follows:



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.<59>

ActualAvailableAllocationUnits (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.<60>

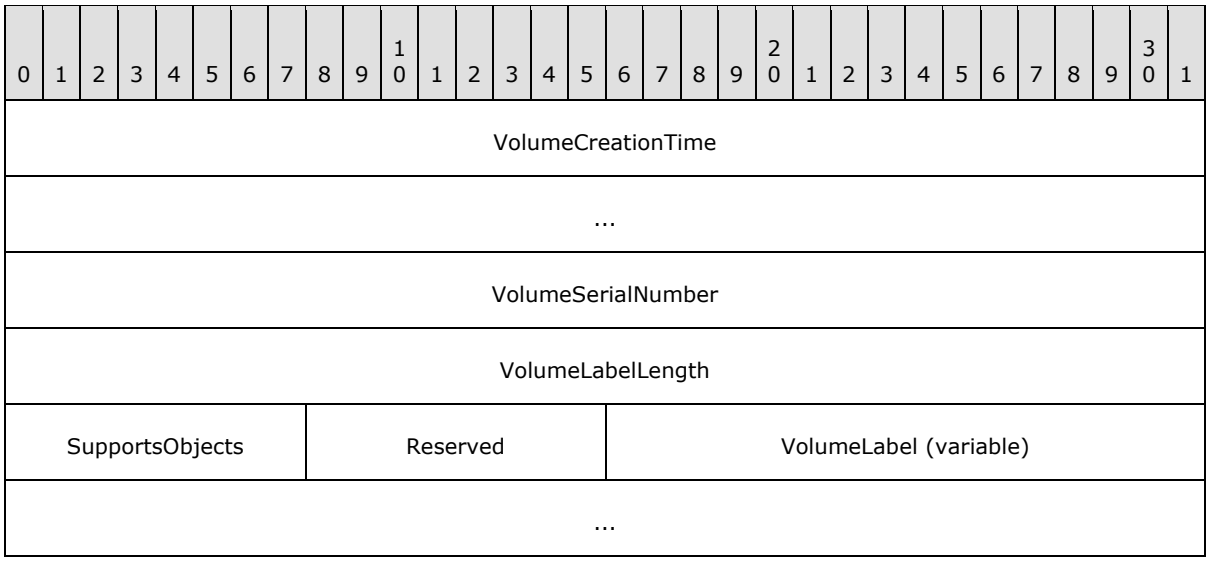
SectorsPerAllocationUnit (4 bytes): A 32-bit unsigned integer that contains the number of sectors in each allocation unit.

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

2.5.8 FileFsVolumeInformation

This information class is used to query information on a volume on which a file system is mounted. The message contains a FILE_FS_VOLUME_INFORMATION data element.

The FILE_FS_VOLUME_INFORMATION data element is as follows:



VolumeCreationTime (8 bytes): A 64-bit signed integer that contains the time when the volume was created in the format of a **FILETIME** structure. 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.

SupportsObjects (1 byte): A 1-byte Boolean (unsigned char) that is TRUE (0x01) if the file system supports **object-oriented file system** objects; otherwise, FALSE (0x00).<61>

Value	Meaning
01	TRUE
00	FALSE

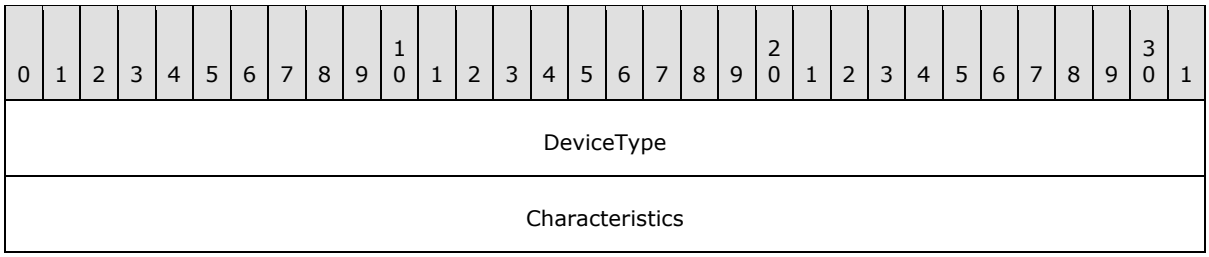
Reserved (1 byte): MUST be ignored by the receiver.

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.

2.5.9 FileFsDeviceInformation

This information class is used to query device information associated with a file system volume. The message contains a FILE_FS_DEVICE_INFORMATION data element.

The FILE_FS_DEVICE_INFORMATION data element is as follows:



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.

Name	Value
FILE_REMOVABLE_MEDIA	0x00000001
FILE_READ_ONLY_DEVICE	0x00000002
FILE_FLOPPY_DISKETTE	0x00000004
FILE_WRITE_ONCE_MEDIA	0x00000008
FILE_REMOTE_DEVICE	0x00000010
FILE_DEVICE_IS_MOUNTED	0x00000020
FILE_VIRTUAL_VOLUME	0x00000040
FILE_AUTOGENERATED_DEVICE_NAME	0x00000080
FILE_DEVICE_SECURE_OPEN	0x00000100
FILE_CHARACTERISTIC_PNP_DEVICE	0x00000800
FILE_CHARACTERISTIC_TS_DEVICE	0x00001000
FILE_CHARACTERISTIC_WEBDAV_DEVICE	0x00002000

2.6 File Attributes

The following attributes are defined by Windows 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 Windows interprets a value of 0x00000000 in the **FileAttributes** field to mean 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 needs to be archived. Applications use this attribute to mark files for backup or removal.
FILE_ATTRIBUTE_ATOMIC_WRITE 0x00000200	This flag has been deprecated.
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 0x00000010	This item is a directory.
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 or directory that does not have other attributes set. This attribute is valid only when no other flags are set.
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.
FILE_ATTRIBUTE_XACTION_WRITE 0x00000400	This flag has been deprecated.

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

The following sections specify security considerations for implementers of File System Control Codes.

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

Security Parameter	Section
FSCTL_ALLOW_EXTENDED_DASD_IO requires the MANAGE_VOLUME_ACCESS privilege	2.3.1

5 Appendix A: Windows Behavior

The information in this specification is applicable to the following versions of Windows:

- Windows NT
- Windows 2000
- Windows XP
- Windows Server 2003
- Windows Vista

Exceptions, if any, are noted below. Unless otherwise specified, any statement of optional behavior in this specification prescribed using the terms SHOULD or SHOULD NOT implies Windows behavior in accordance with the SHOULD or SHOULD NOT prescription. Unless otherwise specified, the term MAY implies that Windows does not follow the prescription.

<1> Section 2.1.4.1: 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.

<2> Section 2.1.4.1: 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).

<3> Section 2.1.4.2: 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).

<4> 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, directory, or volume 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 2.3.48).

<5> 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.

<6> Section 2.3.2: The caller did not have the MANAGE_VOLUME_ACCESS privilege when it opened the handle.

<7> Section 2.3.4: Windows will try 16 times to generate a unique ID, and will fail with this status if 16 attempts have been unsuccessful.

<8> Section 2.3.12: Equivalent to COMPRESSION_FORMAT_LZNT1.

<9> Section 2.3.12: The LZNT1 is the only compression algorithm implemented on Windows 2000, Windows XP, Windows Server 2003, and Windows Vista. Therefore, requests for COMPRESSION_FORMAT_DEFAULT and COMPRESSION_FORMAT_LZNT1 are equivalent from the server perspective.

<10> Section 2.3.12: Windows 2000, Windows XP, Windows Server 2003, and Windows Vista 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.

<11> 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.

<12> 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.

<13> Section 2.3.22: NTFS and FAT on Windows 2000, Windows XP, Windows Server 2003, and Windows Vista ignore the input parameter and return STATUS_SUCCESS whenever this FSCTL is invoked.

<14> Section 2.3.30: 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.

<15> Section 2.3.30: Sparse files are supported by Windows 2000, Windows XP, Windows Server 2003, and Windows Vista. 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.

<16> Section 2.3.30: Windows does not track every piece of zero (0) or nonzero data. Because zero (0) is often a perfectly legal datum, 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.

<17> Section 2.3.32: The major version number is 2 for file systems created on Windows 2000, Windows XP, Windows Server 2003, and Windows Vista.

<18> Section 2.3.32: The minor version number is 0 for file systems created on Windows 2000, Windows XP, Windows Server 2003, and Windows Vista.

<19> Section 2.3.32: 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.

<20> Section 2.3.35: Equivalent to COMPRESSION_FORMAT_LZNT1.

<21> Section 2.3.35: The LZNT1 is the only compression algorithm implemented on Windows 2000, Windows XP, Windows Server 2003, and Windows Vista. Therefore, requests for COMPRESSION_FORMAT_DEFAULT and COMPRESSION_FORMAT_LZNT1 are equivalent from the server's perspective.

<22> Section 2.3.37: 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.

<23> Section 2.3.37: All Windows versions: This request is never sent to a remote server.

<24> Section 2.3.39: 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.

<25> Section 2.3.45: 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.

<26> Section 2.3.47: 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.

<27> Section 2.3.48: If the Microsoft Single-Instance Storage (SIS) filter is not installed on the server, the FSCTL returns STATUS_INVALID_DEVICE_REQUEST.

<28> 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.

<29> 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).

<30> 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.

<31> 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.

<32> 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, and Windows Vista set this value to 0 for files on NTFS file systems.

<33> Section 2.4.9: Windows 2000, Windows XP, Windows Server 2003, and Windows Vista implement only one compression algorithm, LZNT1. For more information, see [UASDC]. COMPRESSION_FORMAT_DEFAULT is therefore equivalent to COMPRESSION_FORMAT_LZNT1.

<34> 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.

<35> Section 2.4.9: NTFS uses a value of 12 for the ChunkShift so that compression chunks are 4 kilobytes in size.

<36> 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-kilobyte bytes. 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.

<37> 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, and Windows Vista set this value to 0 for files on NTFS file systems.

<38> 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.

<39> 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, and Windows Vista set this value to 0 for files on NTFS file systems.

<40> 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 and Windows Vista set this value to 0 for files on NTFS file systems.

<41> 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, and Windows Vista set this value to 0 for files on NTFS file systems.

<42> Section 2.4.25: 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, and Windows Vista set this value to 0 for files on NTFS file systems.

<43> Section 2.4.27: The Microsoft FAT file system does not support the use of ObjectIds, and returns a status code of STATUS_INVALID_DEVICE_REQUEST.

<44> Section 2.4.27: The Microsoft Distributed Link Tracking protocols (see [MS-DLTW] section 3.1.6) use the first type of object ID structure for link tracking.

<45> Section 2.4.27.1: 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.

<46> Section 2.4.38: The FILE_VALID_DATA_LENGTH_INFORMATION structure is used to set a new valid data length for a file on an NTFS volume. 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.

<47> 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.

<48> Section 2.5.1: This attribute is only available on Windows XP, Windows Server 2003, and Windows Vista.

<49> Section 2.5.1: NTFS version 2 is supported on Windows 2000, Windows XP, Windows Server 2003, and Windows Vista.

<50> Section 2.5.2: Setting quota information requires the caller to have permission to open a volume handle or handle to the quota index file for write access.

<51> Section 2.5.2: Windows sets this value to 0.

<52> Section 2.5.2: Windows sets this value to 0.

<53> Section 2.5.2: Windows sets this value to 0.

<54> Section 2.5.2: Windows sets flags not defined below to zero.

<55> Section 2.5.2: Logging makes an entry in the Windows application event log.

<56> Section 2.5.4: In Windows 2000, Windows XP, Windows Server 2003, and Windows Vista, 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.

<57> Section 2.5.4: In Windows 2000, Windows XP, Windows Server 2003, and Windows Vista, if per-user quotas are in use, this value may be less than the total number of free allocation units on the disk.

<58> Section 2.5.6: Windows does not write information into the **ExtendedInfo** field for file systems.

<59> Section 2.5.7: In Windows 2000, Windows XP, Windows Server 2003, and Windows Vista, 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.

<60> Section 2.5.7: In Windows 2000, Windows XP, Windows Server 2003, and Windows Vista, if per-user quotas are in use, this value may be less than the total number of free allocation units on the disk.

<61> Section 2.5.8: This value is TRUE for NTFS and FALSE for other file systems implemented by Windows.

6 Index

G

I

M

N

O

P

R

S

T

V

W