Tell us about your PDF experience.

Open Specifications

Article • 08/01/2023

Through the Open Specifications program, Microsoft is helping developers open new opportunities by making technical documents related to interoperability for certain popular Microsoft products available to view and download at no charge.

Although the Open Specifications technical documents are freely available, many of them include patented inventions. Some of these patents are available at no charge under the Open Specifications Promise or the Microsoft Community Promise. The remaining patents are available through various licensing programs. For more information, please visit the Microsoft Open Specifications Dev Center website or send an email message to the IP Licensing Team.

Expand table

	Protocols Microsoft publishes technical documents for protocols that are implemented in Windows client (including .NET Framework) and Windows Server (collectively Windows), Office, SharePoint Products and Technologies, Exchange Server, and Microsoft SQL Server and are used to communicate with other Microsoft software products.
	Computer Languages Microsoft publishes technical documents for the VBA programming language and Extensible Application Markup Language (XAML).
	Standards Support Microsoft publishes technical documents that describe support for specific standards implemented in Exchange Server and Outlook; Internet Explorer; OData; Microsoft SQL Server; Windows WordPad; and Word, Excel, and PowerPoint.
	Data Portability Microsoft publishes technical documents for the file formats created by Word, Excel, PowerPoint, and Outlook and by SQL Server. Additionally, it publishes technical documents that describe how user-created data in SQL Server can be extracted for use in other software products.

Microsoft revises the technical documents regularly and, particularly, in connection with the release of significant product updates and new versions.

Q&A and Blogs

Expand table

	Exchange Server Open Specifications and Standards Support - Microsoft Q&A Home to technical questions and answers about Exchange Server Open Specifications and Standards Support documents.
	Office Open Specifications, Standards Support, and File Formats - Microsoft Q&A Home to technical questions and answers about Office Open Specifications, Standards Support, and File Format documents.
	SharePoint Server Open Specifications - Microsoft Q&A Home to technical questions and answers about SharePoint Server Open Specifications documents.
	SQL Server Open Specifications, Standards Support and Data Portability - Microsoft Q&A Home to technical questions and answers about SQL Server Open Specifications, Standards Support and Data Portability documents.
	Windows Open Specifications and Standards Support - Microsoft Q&A Home to technical questions and answers about Windows Open Specifications and Standards Support documents.
	Open Specifications Blogs These blogs, authored by the engineers who support the Open Specifications documents, provide a different venue for further discussion of those documents.

Also in this section

- Programs

Protocols

Article • 08/01/2023

The Protocols section provides detailed Open Specifications technical documents for certain protocols that are implemented in Exchange Server, Office, SharePoint Products and Technologies, Microsoft SQL Server, Windows client (including .NET Framework) and Windows Server collectively published under Windows Protocols and are used to communicate with other Microsoft software products.

Although the Open Specifications technical documents are freely available, many of them include patented inventions. Some of these patents are available at no charge under the Open Specifications Promise or the Microsoft Community Promise. The remaining patents are available through various licensing programs. For more information, please visit the Microsoft Open Specifications Dev Center website or send an email message to the IP Licensing Team.

Q&A and Blogs

	Exchange Server Open Specifications and Standards Support - Microsoft Q&A Home to technical questions and answers about Exchange Server Open Specifications and Standards Support documents.
	Office Open Specifications, Standards Support, and File Formats - Microsoft Q&A Home to technical questions and answers about Office Open Specifications, Standards Support, and File Format documents.
	SharePoint Server Open Specifications - Microsoft Q&A Home to technical questions and answers about SharePoint Server Open Specifications documents.
	SQL Server Open Specifications, Standards Support and Data Portability - Microsoft Q&A Home to technical questions and answers about SQL Server Open Specifications, Standards Support and Data Portability documents.
	Windows Open Specifications and Standards Support - Microsoft Q&A Home to technical questions and answers about Windows Open Specifications and Standards Support documents.
	Open Specifications Blogs

	These blogs, authored by the engineers who support the Open Specifications documents, provide a different venue for further discussion of those documents.

Explore the Protocols Documentation

- Exchange Server Protocols
- Office Protocols
- SharePoint Products and Technologies Protocols
- Microsoft SQL Server Protocols
- Windows Protocols

Downloads	Related Sections	Sites
Exchange Server Protocols .zip file (120+ MB) Microsoft Office Protocols .zip file (80+ MB) SharePoint Products and Technologies Protocols .zip file (200+ MB) Microsoft SQL Server Protocols .zip file (50+ MB) Windows Protocols .zip file (500+ MB)	Document Programs Computer Languages Standards Support Data Portability	Open Specifications Dev Center

Windows Protocols

Article04/07/2025

This page and associated content may be updated frequently. We recommend you subscribe to the RSS feed to receive update notifications.

This documentation contains detailed technical specifications for Microsoft protocols that are implemented and used by Windows to interoperate or communicate with other Microsoft products. It also contains technical specifications for extensions to industry-standard and other published protocols that are used by Windows. In addition, the documentation includes a set of companion technology overview and reference documents that supplement the technical specifications with conceptual background, overviews of inter-protocol relationships and interactions, and technical reference information.

Although the Open Specifications technical documents are freely available, many of them include patented inventions. Some of these patents are available at no charge under the Open Specifications Promise or the Microsoft Community Promise. The remaining patents are available through various licensing programs. For more information, please visit the Microsoft Open Specifications Dev Center or send an email message to the IP Licensing Team.

Explore Windows Protocols Documentation

Expand table

	What's New and Changed in Windows Documents Provides information about and links to new and updated protocol documents that contain details about the most recently released versions of Windows Client and Windows Server operating systems.
	Windows Protocols Preview Documents Provides preliminary versions of new or updated Open Specifications technical documents for community review and feedback.
	Windows Protocols Errata Provides clarifications and adjustments of content issues in published versions of protocol documents that could impact an implementation.

	What's New and Changed in Windows Documents Provides information about and links to new and updated protocol documents that contain details about the most recently released versions of Windows Client and Windows Server operating systems.
	Windows Protocols Overview Documents Provides information about the protocols and other technologies that are included in the Windows Protocols documentation set and the relationships among those technologies.
	Windows Protocols Technical Documents Provides detailed technical specifications for Microsoft proprietary protocols (including extensions to industry-standard or other published protocols) and other technologies that are used by Windows to communicate with other Microsoft products.
	Windows Protocols Reference Documents Provides reference documents that are available for use with the Windows Protocols documentation set.

Q&A and Blogs

Expand table

	Windows Open Specifications and Standards Support - Microsoft Q&A Home to technical questions and answers about Windows Open Specifications and Standards Support documents.
	Open Specifications Blogs These blogs, authored by the engineers who support the Open Specifications documents, provide a different venue for further discussion of those documents.
	Windows Interoperability Developer Blog This blog is updated with the latest information on releases of the Open Specifications documents.

Expand table

Downloads	Sites
Windows Protocols PDF .zip file (600+ MB)	Open Specifications Developer Center FAQ: Network Captures for Technology Overview Documents

Downloads	Sites
Standards Support Downloads	Windows Protocols Archived Forums

Technical Documents

Article03/13/2023

This section provides information about the technical specifications that are contained in the Windows Protocols documentation set.

For preview or pre-release versions of the technical specifications, see Preview Documents.

Note The inter-document links in a PDF version of a technical specification document are functional only if all the cross-referenced documents are saved to the same local directory folder. An error message appears if you click a link that references a PDF document that is not located in the same folder (when viewing via your local hard drive) or is part of a different download (when viewing online). To save a complete set of PDF files to the same folder, download the Windows Protocols .zip file. This is a large file and can take a few minutes to download.

Expand table

Specification	Description
[MC-BUP]: Background Intelligent Transfer Service (BITS) Upload Protocol	Specifies the Background Intelligent Transfer Service (BITS) Upload Protocol, which is used to upload large entities from a client to a server over networks with frequent disconnections, and to send notifications from the server to a server application about the availability of the uploaded entities. Click here to view this version of the [MC-BUP] PDF.
[MC-CCFG]: Server Cluster: Configuration (ClusCfg) Protocol	Specifies the Server Cluster: Configuration (ClusCfg) Protocol, which enables users to restore a node that is no longer a configured member of a failover cluster back to its pre-cluster installation state. Click here to view this version of the [MC-CCFG] PDF.
[MC-COMQC]: Component Object Model Plus (COM+) Queued Components Protocol	Specifies the Component Object Model Plus (COM+) Queued Components Protocol, which is used for persisting method calls made on COM+ objects in such a way that they can later be played back and executed. Click here to view this version of the [MC-COMQC] PDF.
[MC-CSDL]: Conceptual Schema Definition File Format	Specifies the Conceptual Schema Definition File Format, which defines some well-known primitive types, such as Edm.String, that are used as the building blocks for structural types like Entity Types and Complex Types.

Specification	Description
	Click here to view this version of the [MC-CSDL] PDF.
[MC-DPL4CS]: DirectPlay 4 Protocol: Core and Service Providers	Specifies DirectPlay 4 Protocol: Core and Service Providers. This protocol enables the implementation of functions to enumerate hosted game sessions and players, to add and remove game players, and to interchange data between game instances. Click here to view this version of the [MC-DPL4CS] PDF.
[MC-DPL4R]: DirectPlay 4 Protocol: Reliable	Specifies the DirectPlay 4 Protocol: Reliable, which describes functionality related to the reliable delivery of DirectPlay 4 messages and provides throttling for applications that use DirectPlay 4. Click here to view this version of the [MC-DPL4R] PDF.
[MC-DPL8CS]: DirectPlay 8 Protocol: Core and Service Providers	Specifies the DirectPlay 8 Protocol: Core and Service Providers, which creates and manages game sessions over existing datagram protocols such as UDP. Click here to view this version of the [MC-DPL8CS] PDF.
[MC-DPL8R]: DirectPlay 8 Protocol: Reliable	Specifies the DirectPlay 8 Protocol: Reliable, which provides mixed, not reliable, and reliable messages over existing datagram protocols such as the User Datagram Protocol (UDP). Click here to view this version of the [MC-DPL8R] PDF.
[MC-DPLHP]: DirectPlay 8 Protocol: Host and Port Enumeration	Specifies the DirectPlay 8 Protocol: Host and Port Enumeration, which enables a DirectPlay 8 client application to discover one or more DirectPlay 8 server applications. Click here to view this version of the [MC-DPLHP] PDF.
[MC-DPLNAT]: DirectPlay 8 Protocol: NAT Locator	Specifies the DirectPlay 8 Protocol: NAT Locator, which provides extensions to the DirectPlay 8 Core and Service Providers Protocol (as specified in [MC-DPL8CS]) to improve Network Address Translation (NAT) support. Click here to view this version of the [MC-DPLNAT] PDF.
[MC-DPLVP]: DirectPlay Voice Protocol	Specifies the DirectPlay Voice Protocol, which is used to provide voice communications for applications that use the DirectPlay protocol to communicate. Click here to view this version of the [MC-DPLVP] PDF.
[MC-DRT]: Distributed Routing Table (DRT) Version 1.0	Specifies the Distributed Routing Table (DRT) Version 1.0 protocol, which is used to maintain a network of nodes (referred to as a cloud) and to resolve keys to their endpoint information when requested by a node within the cloud.

Specification	Description
	Click here to view this version of the [MC-DRT] PDF.
[MC-DTCXA]: MSDTC Connection Manager: OleTx XA Protocol	Specifies the MSDTC Connection Manager: OleTx Transaction Protocol, which describes the extensions that support XA [XOPEN-DTP]-compliant software components in an OleTx distributed transaction processing environment. Click here to view this version of the [MC-DTCXA] PDF.
[MC-EDMX]: Entity Data Model for Data Services Packaging Format	Specifies the Entity Data Model for Data Services Packaging Format (EDMX), which is an XML-based file format that serves as the packaging format for the service metadata of a data service. Click here to view this version of the [MC-EDMX] PDF.
[MC-IISA]: Internet Information Services (IIS) Application Host COM Protocol	Specifies the Internet Information Services (IIS) Application Host COM Protocol, which provides read/write access to administrative configuration data that is located on a remote server. Click here to view this version of the [MC-IISA] PDF.
[MC-MQAC]: Message Queuing (MSMQ): ActiveX Client Protocol	Specifies the Message Queuing (MSMQ): ActiveX Client Protocol, which is a collection of Distributed Component Object Model (DCOM) [MS-DCOM] interfaces that expose message queuing functionality for use by client applications. Click here to view this version of the [MC-MQAC] PDF.
[MC-MQSRM]: Message Queuing (MSMQ): SOAP Reliable Messaging Protocol (SRMP)	Specifies the Message Queuing (MSMQ): SOAP Reliable Messaging Protocol (SRMP), which defines a mechanism for reliably transferring messages between two message queues that are located on two different hosts. Click here to view this version of the [MC-MQSRM] PDF.
[MC-NBFS]: .NET Binary Format: SOAP Data Structure	Specifies the SOAP data structure for the .NET Binary Format for XML. This structure uses the XML data structure format [MC-NBFX], but specifies the set of strings to which a producer and consumer can refer. Click here to view this version of the [MC-NBFS] PDF.
[MC-NBFSE]: .NET Binary Format: SOAP Extension	Specifies the SOAP extension for the .NET Binary Format for XML. This SOAP extension is a new format built by extending the format specified in [MC-NBFS]; it provides a context under which strings may be transmitted once and referred to by subsequent documents in order to reduce the size of the documents. Click here to view this version of the [MC-NBFSE] PDF.

Specification	Description
[MC-NBFX]: .NET Binary Format: XML Data Structure	Specifies the XML data structure for the .NET Binary Format for XML. This format can represent many XML documents, as specified in [XML1.0]. The purpose of the format is to reduce the processing costs associated with XML documents by encoding an XML document in fewer bytes than the same document encoded in UTF-8, as specified in [RFC2279]. Click here to view this version of the [MC-NBFX] PDF.
[MC-NETCEX]: .NET Context Exchange Protocol	Specifies the .NET Context Exchange Protocol, which defines a message syntax for identifying context that is shared between a client and a server, and a protocol for establishing that context. Click here to view this version of the [MC-NETCEX] PDF.
[MC-NMF]: .NET Message Framing Protocol	Specifies the .NET Message Framing Protocol, which defines a mechanism for framing messages. While this is primarily aimed at framing SOAP messages, the protocol can be used to frame other message types as well. Click here to view this version of the [MC-NMF] PDF.
[MC-NPR]: .NET Packet Routing Protocol	Specifies the .NET Packet Routing Protocol, which defines a SOAP message header to indicate that a message can safely be treated as a packet or datagram. Click here to view this version of the [MC-NPR] PDF.
[MC-PRCH]: Peer Channel Protocol	Specifies the Peer Channel Protocol, which is used for broadcasting messages over a virtual network of cooperating nodes. Click here to view this version of the [MC-PRCH] PDF.
[MC-PRCR]: Peer Channel Custom Resolver Protocol	Specifies the Peer Channel Custom Resolver Protocol, which is used for storage and retrieval of endpoint information of clients with access to a known service. Click here to view this version of the [MC-PRCR] PDF.
[MC-SMP]: Session Multiplex Protocol	Specifies the Session Multiplex Protocol, which provides session management capabilities between a database client and a database server. This protocol enables multiple logical client connections to connect to a single server over a single physical connection. Click here to view this version of the [MC-SMP] PDF.
[MC-SQLR]: SQL Server Resolution Protocol	Specifies the SQL Server Resolution Protocol, which facilitates connectivity to a database server. Click here to view this version of the [MC-SQLR] PDF.

Specification	Description
[MS-ABTP]: Automatic Bluetooth Pairing Protocol	Specifies the Automatic Bluetooth Pairing Protocol, which facilitates the establishment of a secure, trusted Bluetooth pairing relationship between two devices without requiring any user interaction at the time of pairing. Click here to view this version of the [MS-ABTP] PDF.
[MS-ADA1]: Active Directory Schema Attributes A-L	Specifies the Active Directory Schema Attributes A-L, which contains a partial list of the objects that exist in the Active Directory schema (attributes beginning with A - L). Click here to view this version of the [MS-ADA1] PDF.
[MS-ADA2]: Active Directory Schema Attributes M	Specifies the Active Directory Schema Attributes M, which contains a partial list of the objects that exist in the Active Directory schema (attributes beginning with M). Click here to view this version of the [MS-ADA2] PDF.
[MS-ADA3]: Active Directory Schema Attributes N-Z	Specifies the Active Directory Schema Attributes N-Z, which contains a partial list of the objects that exist in the Active Directory schema (attributes beginning with N through Z). Click here to view this version of the [MS-ADA3] PDF.
[MS-ADCAP]: Active Directory Web Services: Custom Action Protocol	Specifies the Active Directory Web Services: Custom Action Protocol, used for directory access in identity management and topology management. This protocol enables the transition of client applications that are currently using non-web services protocols for managing information held in directory services to instead use Web services protocols. Click here to view this version of the [MS-ADCAP] PDF.
[MS-ADDM]: Active Directory Web Services: Data Model and Common Elements	Specifies the Active Directory Web Services: Data Model and Common Elements. This protocol contains an XML data model and other protocol components (such as the definition of an XPath 1.0-derived selection language) that are used in various protocols that belong to the set of Active Directory Web Services protocols. Click here to view this version of the [MS-ADDM] PDF.
[MS-ADFSOAL]: Active Directory Federation Services OAuth Authorization Code Lookup Protocol	Specifies the Active Directory Federation Services OAuth Authorization Code Lookup Protocol, which is used to find the issuing server of an access token for an OAuth authorization code. Click here to view this version of the [MS-ADFSOAL] PDF.
[MS-ADFSPIP]: Active Directory Federation	Specifies the Active Directory Federation Services Proxy and Web Application Proxy Integration Protocol. This protocol integrates

Specification	Description
Services and Proxy Integration Protocol	Active Directory Federation Services with an authentication and application proxy to enable access to services located inside the boundaries of the corporate network for clients that are located outside of that boundary. Click here to view this version of the [MS-ADFSPIP] PDF.
[MS-ADFSPP]: Active Directory Federation Service (AD FS) Proxy Protocol	Specifies the Federation Service Proxy Protocol, which is used by a security token service (STS) proxy to obtain configuration data about an STS in order to assist users in selecting an acceptable security realm from which to obtain a security token. Click here to view this version of the [MS-ADFSPP] PDF.
[MS-ADFSWAP]: Active Directory Federation Service (AD FS) Web Agent Protocol	Specifies the Federation Service Web Agent Protocol, which is used by a Web service (WS) resource to obtain configuration data about a security token service (STS) in order to validate tokens from that STS using the protocol defined in [MS-MWBF]. Click here to view this version of the [MS-ADFSWAP] PDF.
[MS-ADLS]: Active Directory Lightweight Directory Services Schema	Specifies the Active Directory Lightweight Directory Services Schema, which contains a list of the objects that exist in the Active Directory Lightweight Directory Services schema. Click here to view this version of the [MS-ADLS] PDF.
[MS-ADSC]: Active Directory Schema Classes	Specifies the Active Directory Schema Classes, which contains a partial list of objects that exist in the Active Directory schema. Click here to view this version of the [MS-ADSC] PDF.
[MS-ADTG]: Remote Data Services (RDS) Transport Protocol	Specifies the Remote Data Services (RDS) Transport Protocol, an HTTP request/response protocol that facilitates remote method definition and invocation, method definitions for executing database commands and for synchronizing database results, and definition of a record format for encoding of database results. Click here to view this version of the [MS-ADTG] PDF.
[MS-ADTS]: Active Directory Technical Specification	Specifies the core functionality of Active Directory. Active Directory extends and provides variations of the Lightweight Directory Access Protocol (LDAP). Click here to view this version of the [MS-ADTS] PDF.
[MS-AIPS]: Authenticated Internet Protocol	Specifies the Authenticated Internet Protocol. This protocol supports a more generalized authentication exchange than the Internet Key Exchange Protocol and provides the optimizations in key exchange and policy discoverability.

Specification	Description
	Click here to view this version of the [MS-AIPS] PDF.
[MS-APDS]: Authentication Protocol Domain Support	Specifies Authentication Protocol Domain Support, which is the communication process between a server and a domain controller that uses Netlogon interfaces to complete an authentication sequence. Click here to view this version of the [MS-APDS] PDF.
[MS-ASP]: ASP.NET State Server Protocol	Specifies the ASP.NET State Server Protocol, which is a contract for transmitting session state data between a client and a state server. Click here to view this version of the [MS-ASP] PDF.
[MS-AZMP]: Authorization Manager (AzMan) Policy File Format	Specifies the Authorization Manager (AzMan) Policy File Format, which defines the XML structure of AzMan policy files. These files are used by the Microsoft Management Console (MMC) AzMan snap-in and the authorization manager runtime. Click here to view this version of the [MS-AZMP] PDF.
[MS-BDSRR]: Business Document Scanning: Scan Repository Capabilities and Status Retrieval Protocol	Specifies the Business Document Scanning: Scan Repository Capabilities and Status Retrieval Protocol, which is used to query a server for the capabilities and status of the scan repository. Click here to view this version of the [MS-BDSRR] PDF.
[MS-BGPP]: Border Gateway Protocol (BGP) Profile	Specifies Border Gateway Protocol (BGP) Profile a dynamic routing protocol that automatically learns routes between sites that are connected using site-to-site VPN connections and clarifies what portions of [RFC1997] and [RFC4271] are not supported. Click here to view this version of the [MS-BGPP] PDF.
[MS-BKRP]: BackupKey Remote Protocol	Specifies the BackupKey Remote Protocol. This protocol encrypts secret values (such as cryptographic keys) so they can be backed up to storage that is not specially protected, and enables decryption of such values if recovery is necessary. Click here to view this version of the [MS-BKRP] PDF.
[MS-BKUP]: Microsoft NT Backup File Structure	Specifies the Microsoft NT Backup File Structure protocol, which describes the network format of the Windows NT backup file format and its constituent structures that may be used in other protocols. Click here to view this version of the [MS-BKUP] PDF.

Specification	Description
[MS-BPAU]: Background Intelligent Transfer Service (BITS) Peer-Caching: Peer Authentication Protocol	Specifies the Background Intelligent Transfer Service (BITS) Peer-Caching: Peer Authentication Protocol. This protocol provides authentication for computers in an Active Directory domain in support of the BITS Peer-Caching Content Retrieval Protocol ([MS-BPCR]). Click here to view this version of the [MS-BPAU] PDF.
[MS-BPCR]: Background Intelligent Transfer Service (BITS) Peer-Caching: Content Retrieval Protocol	Specifies the Background Intelligent Transfer Service (BITS) Peer-Caching: Content Retrieval Protocol, which is one of the family of protocols that implements a distributed URL cache known as ""BITS peer-caching"". Other protocols in the family are used to discover potential peers and to authenticate them. Click here to view this version of the [MS-BPCR] PDF.
[MS-BPDP]: Background Intelligent Transfer Service (BITS) Peer-Caching: Peer Discovery Protocol	Specifies the Background Intelligent Transfer Service (BITS) Peer-Caching: Peer Discovery Protocol, which is used to locate hosts in a domain that supports the URL-caching protocol implemented by BITS. Click here to view this version of the [MS-BPDP] PDF.
[MS-BRWS]: Common Internet File System (CIFS) Browser Protocol	Specifies the Common Internet File System (CIFS) Browser Protocol, which updates all backup browser servers with the contents of the response to a NetServerEnum2 request and shares the processing load of enumerating the services available in the network across different servers. Click here to view this version of the [MS-BRWS] PDF.
[MS-BRWSA]: Common Internet File System (CIFS) Browser Auxiliary Protocol	Specifies the Common Internet File System (CIFS) Browser Auxiliary Protocol, which is used by the master browser server to query configuration information for the domains from the domain master browser server. Click here to view this version of the [MS-BRWSA] PDF.
[MS-CAPR]: Central Access Policy Identifier (ID) Retrieval Protocol	Specifies the Central Access Policy ID Retrieval Protocol, which allows administrative applications to retrieve the set of central access policies deployed on remote computers. Click here to view this version of the [MS-CAPR] PDF.
[MS-CBCP]: Callback Control Protocol	Specifies the Callback Control Protocol, which provides a standard method for transporting multi-protocol datagrams over point-to-point links. Click here to view this version of the [MS-CBCP] PDF.

Specification	Description
[MS-CDP]: Connected Devices Platform Protocol Version 3	Specifies the Connected Devices Platform Protocol Version 3. This protocol provides a discovery system to authenticate and verify users and devices, as well as providing a message exchange between devices. It provides a transport-agnostic means of building connections among all of a user's devices, whether available through the cloud or through direct physical presence. Click here to view this version of the [MS-CDP] PDF.
[MS-CER]: Corporate Error Reporting Version 1.0 Protocol	Specifies the Corporate Error Reporting Version 1.0 Protocol, which enables an organization to copy error reports from a set of client machines to a CER file share on a specified Server Message Block (SMB) Protocol file server with additional configuration options. Click here to view this version of the [MS-CER] PDF.
[MS-CER2]: Corporate Error Reporting V.2 Protocol	Specifies the Corporate Error Reporting V.2 Protocol, which enables enterprise computing sites to manage all error reporting information within the organization. Click here to view this version of the [MS-CER2] PDF.
[MS-CFB]: Compound File Binary File Format	Specifies the Compound File Binary File Format, a general-purpose file format that provides a file-system-like structure within a file for the storage of arbitrary, application-specific streams of data. Click here to view this version of the [MS-CFB] PDF.
[MS-CHAP]: Extensible Authentication Protocol Method for Microsoft Challenge Handshake Authentication Protocol (CHAP)	Specifies the Extensible Authentication Protocol Method for Microsoft Challenge Handshake Authentication Protocol (CHAP). This protocol enables extensible authentication for network access. Click here to view this version of the [MS-CHAP] PDF.
[MS-CIFS]: Common Internet File System (CIFS) Protocol	Specifies the Common Internet File System (CIFS) Protocol, a cross-platform, transport-independent protocol that provides a mechanism for client systems to use file and print services made available by server systems over a network. Click here to view this version of the [MS-CIFS] PDF.
[MS-CMOM]: MSDTC Connection Manager: OleTx Management Protocol	Specifies the MSDTC Connection Manager: OleTx Management Protocol. This protocol enables the remote management of an OleTx Transaction Manager and its extensions. Click here to view this version of the [MS-CMOM] PDF.
[MS-CMP]: MSDTC Connection Manager:	Specifies the MSDTC Connection Manager Protocol: Connection Multiplexing Protocol, which enables partners to multiplex any

Specification	Description
OleTx Multiplexing Protocol	number of two-way connections over the MSDTC Connection Manager: OleTx Transports Protocol session. Click here to view this version of the [MS-CMP] PDF.
[MS-CMPO]: MSDTC Connection Manager: OleTx Transports Protocol	Specifies the MSDTC Connection Manager: OleTx Transports Protocol, a peer-to-peer messaging protocol layered over a bidirectional pair of RPC connections. Click here to view this version of the [MS-CMPO] PDF.
[MS-CMRP]: Failover Cluster: Management API (ClusAPI) Protocol	Specifies the Failover Cluster: Management API (ClusAPI) Protocol, an RPC-based protocol that is used for remotely managing a cluster. Click here to view this version of the [MS-CMRP] PDF.
[MS-COM]: Component Object Model Plus (COM+) Protocol	Specifies the Component Object Model Plus (COM+) Protocol, which consists of a DCOM interface (and DCOM protocol extensions) that is used for adding transactions, implementing synchronization, managing multiple object class configurations, enforcing security, and providing additional functionality and attributes to DCOM-based distributed object applications. Click here to view this version of the [MS-COM] PDF.
[MS-COMA]: Component Object Model Plus (COM+) Remote Administration Protocol	Specifies the Component Object Model Plus (COM+) Remote Administration Protocol, which enables remote clients to register, import, remove, configure, control, and monitor components and conglomerations for an Object Request Broker (ORB). Click here to view this version of the [MS-COMA] PDF.
[MS-COMEV]: Component Object Model Plus (COM+) Event System Protocol	Specifies the Component Object Model Plus (COM+) Event System Protocol, which is a protocol that exposes DCOM interfaces for storing and managing configuration data for publishers of events and their respective subscribers on remote computers. This protocol also specifies how to get specific information about a publisher and its subscribers. Click here to view this version of the [MS-COMEV] PDF.
[MS-COMT]: Component Object Model Plus (COM+) Tracker Service Protocol	Specifies the Component Object Model Plus (COM+) Tracker Service Protocol, which enables clients to monitor running instances of components. Click here to view this version of the [MS-COMT] PDF.

Specification	Description
[MS-CPSP]: Connection Point Services: Phonebook Data Structure	Specifies the Connection Point Services: Phonebook Data Structure. This structure describes a format for documenting POP entry information and a logical grouping of POPs based on their geographic location. Click here to view this version of the [MS-CPSP] PDF.
[MS-CRTD]: Certificate Templates Structure	Specifies the Certificate Templates Structure. This structure describes the syntax and interpretation of certificate templates, which forms the basis of certificate management for the Certificate Templates Protocol. Click here to view this version of the [MS-CRTD] PDF.
[MS-CSRA]: Certificate Services Remote Administration Protocol	Specifies the Certificate Services Remote Administration Protocol, which consists of a set of Distributed Component Object Model (DCOM) interfaces that enable administrative tools to configure the state and policy of a certification authority (CA) on a server. Click here to view this version of the [MS-CSRA] PDF.
[MS-CSSP]: Credential Security Support Provider (CredSSP) Protocol	Specifies the Credential Security Support Provider (CredSSP) Protocol, which enables an application to securely delegate a user's credentials from a client to a target server. Click here to view this version of the [MS-CSSP] PDF.
[MS-CSVP]: Failover Cluster: Setup and Validation Protocol (ClusPrep)	Specifies the Failover Cluster: Setup and Validation Protocol (ClusPrep), which remotely configures cluster nodes, cleans up cluster nodes, and validates that hardware and software settings are compatible with Failover Clustering. Click here to view this version of the [MS-CSVP] PDF.
[MS-CTA]: Claims Transformation Algorithm	Specifies the Claims Transformation Algorithm (CTA), which consists of two components: a grammar describing a transformation rules language and an algorithm for transforming input claims into output claims. A claim is an assertion about a user identity in the form of a name-value tuple. Sets of claims are transformed from sending authority formats to receiving authority formats at authentication trust traversal boundaries. Click here to view this version of the [MS-CTA] PDF.
[MS-DCOM]: Distributed Component Object Model (DCOM) Remote Protocol	Specifies the Distributed Component Object Model (DCOM) Remote Protocol, which exposes application objects via remote procedure calls (RPCs) and consists of a set of extensions layered on the Microsoft Remote Procedure Call Extensions. Click here to view this version of the [MS-DCOM] PDF.

Specification	Description
[MS-DFSC]: Distributed File System (DFS): Referral Protocol	Specifies the Distributed File System (DFS): Referral Protocol, which enables file system clients to resolve names from a namespace distributed across many servers and geographies into local names on specific file servers. Click here to view this version of the [MS-DFSC] PDF.
[MS-DFSNM]: Distributed File System (DFS): Namespace Management Protocol	Specifies the Distributed File System (DFS): Namespace Management Protocol, which provides an RPC interface for administering DFS configurations. The client is an application that issues method calls on the RPC interface to administer DFS. The server is a DFS service that implements support for this RPC interface for administering DFS. Click here to view this version of the [MS-DFSNM] PDF.
[MS-DFSRH]: DFS Replication Helper Protocol	Specifies the DFS Replication Helper Protocol, which is made up of a set of distributed component object model (DCOM) interfaces for configuring and monitoring DFS Replication Helper Protocols on a server. Click here to view this version of the [MS-DFSRH] PDF.
[MS-DHA]: Device Health Attestation Protocol	Specifies the Device Health Attestation Service Protocol, which enables the assessment of the attested boot state of devices. The outcome of the health assessment is included in a signed health certificate which can then be evaluated by other services to determine whether a device is meeting enterprise corporate policy for device health, or by third party services to identify jailbroken devices that should not receive content or access to certain resources. Click here to view this version of the [MS-DHA] PDF.
[MS-DHCPE]: Dynamic Host Configuration Protocol (DHCP) Extensions	Specifies the Dynamic Host Configuration Protocol (DHCP), which describes the Microsoft specific vendor-class options included in the Microsoft implementation of DHCP. Click here to view this version of the [MS-DHCPE] PDF.
[MS-DHCPF]: DHCP Failover Protocol Extension	Specifies the DHCP Failover Protocol Extension, which extends the DHCP Failover Protocol by encrypting messages sent between the servers in a failover relationship and by providing client implementation options. Click here to view this version of the [MS-DHCPF] PDF.
[MS-DHCPM]: Microsoft Dynamic Host Configuration Protocol	Specifies the Microsoft Dynamic Host Configuration Protocol (DHCP) Server Management Protocol, which defines the RPC interfaces that provide methods for remotely accessing and

Specification	Description
(DHCP) Server Management Protocol	administering the DHCP server. This protocol is a client and server protocol based on RPC that is used in the configuration, management, and monitoring of a DHCP server. Click here to view this version of the [MS-DHCPM] PDF.
[MS-DLNHND]: Digital Living Network Alliance (DLNA) Networked Device Interoperability Guidelines: Microsoft Extensions	Specifies Digital Living Network Alliance (DLNA) Home Networked Device Interoperability Guidelines: Microsoft Extensions. The DLNA Guidelines define protocol extensions to protocols related to streaming of content. Click here to view this version of the [MS-DLNHND] PDF.
[MS-DLTW]: Distributed Link Tracking: Workstation Protocol	Specifies the Distributed Link Tracking: Workstation Protocol, which works with the Distributed Link Tracking (DLT) Central Manager Protocol to discover the new location of a file that has moved. DLT can determine whether the file has moved on a mass-storage device, within a computer, or between computers in a network. Click here to view this version of the [MS-DLTW] PDF.
[MS-DNSP]: Domain Name Service (DNS) Server Management Protocol	Specifies the Domain Name Service (DNS) Server Management Protocol, which defines the RPC interfaces that provide methods for remotely accessing and administering a DNS server. It is a client and server protocol based on RPC that is used in the configuration, management, and monitoring of a DNS server. Click here to view this version of the [MS-DNSP] PDF.
[MS-DPDX]: DirectPlay DXDiag Usage Protocol	Specifies the DirectPlay DXDiag Usage Protocol, intended for peer-to-peer network video gaming and used by the DXDiag application. Click here to view this version of the [MS-DPDX] PDF.
[MS-DPSP]: Digest Protocol Extensions	Specifies the Digest Protocol Extensions, which describes the variations in the Windows implementation of the Digest Authentication protocol from the standard, as specified in [RFC2617]. Click here to view this version of the [MS-DPSP] PDF.
[MS-DPWSSN]: Devices Profile for Web Services (DPWS): Size Negotiation Extension	Specifies the Devices Profile for Web Services (DPWS): Size Negotiation Extension. This is an extension to the Devices Profile for Web Services (DPWS) and enables the negotiation of message sizes between a client and a service for a specific message transaction. Click here to view this version of the [MS-DPWSSN] PDF.

Specification	Description
[MS-DRSR]: Directory Replication Service (DRS) Remote Protocol	Specifies the Directory Replication Service (DRS) Remote Protocol, an RPC protocol for replication and management of data in Active Directory. Click here to view this version of the [MS-DRSR] PDF.
[MS-DSCPM]: Desired State Configuration Pull Model Protocol	Specifies the Desired State Configuration Pull Model Protocol, which is used to get a client's configuration and modules from the server and to report the client's status back to the server. The protocol depends on HTTP for the transfer of all protocol messages. With the Desired State Configuration Pull Model Protocol, binary data flows from the server to the client. Click here to view this version of the [MS-DSCPM] PDF.
[MS-DSLR]: Device Services Lightweight Remoting Protocol	Specifies the Device Services Lightweight Remoting Protocol, which enables remoting of services (objects, function calls, events, and so on) over a reliable point-to-point channel. Click here to view this version of the [MS-DSLR] PDF.
[MS-DSML]: Directory Services Markup Language (DSML) 2.0 Protocol Extensions	Specifies the Directory Services Markup Language (DSML) 2.0 Protocol Extensions. The SOAP session extensions (SSE) make it possible to maintain state information across multiple request/response operations. Click here to view this version of the [MS-DSML] PDF.
[MS-DSSP]: Directory Services Setup Remote Protocol	Specifies the Directory Services Setup Remote Protocol, which exposes an RPC interface that a client can call to obtain domain-related computer state and configuration information. Click here to view this version of the [MS-DSSP] PDF.
[MS-DTCLU]: MSDTC Connection Manager: OleTx Transaction Protocol Logical Unit Mainframe Extension	Specifies the MSDTC Connection Manager: OleTx Transaction Protocol Logical Unit Mainframe Extension, which provides concrete mechanisms for associating an Atomic Transaction and an LU type 6.2 Logical Unit of Work. Click here to view this version of the [MS-DTCLU] PDF.
[MS-DTCM]: MSDTC Connection Manager: OleTx Transaction Internet Protocol	Specifies the MSDTC Connection Manager: OleTx Transaction Internet Protocol, which extends the OleTx protocol (see [MS-DTCO]) to enable its interoperation with the open-standard Transaction Internet Protocol (TIP). Click here to view this version of the [MS-DTCM] PDF.
[MS-DTCO]: MSDTC Connection Manager:	Specifies the MSDTC Connection Manager: OleTx Transaction Protocol, which provides concrete mechanisms for beginning,

Specification	Description
OleTx Transaction Protocol	propagating, and completing atomic transactions. This protocol also provides mechanisms for coordinating agreement on a single atomic outcome for each transaction, and for reliably distributing that outcome to all participants in the transaction. Click here to view this version of the [MS-DTCO] PDF.
[MS-DVRD]: Device Registration Discovery Protocol	Specifies the Device Registration Discovery Protocol, which is used to discover information about servers that can register corporate-owned and personal devices with a corporate network. Click here to view this version of the [MS-DVRD] PDF.
[MS-DVRE]: Device Registration Enrollment Protocol	Specifies the Device Registration Enrollment Protocol, which is used to register corporate-owned and personal devices with a corporate network. Click here to view this version of the [MS-DVRE] PDF.
[MS-DVRJ]: Device Registration Join Protocol	Specifies the Device Registration Join Protocol, which establishes a device identity between the physical device and a directory service. The identity is used at the system level to identify the device only. Click here to view this version of the [MS-DVRJ] PDF.
[MS-ECS]: Enterprise Client Synchronization Protocol	Specifies the Enterprise Client Sync protocol, which enables devices (such as tablets, PCs, or laptops) to synchronize files to and from a file server in a REST-based manner. Click here to view this version of the [MS-ECS] PDF.
[MS-EERR]: ExtendedError Remote Data Structure	Specifies the ExtendedError Remote Data Structure, which encodes extended error information. This data structure assumes that the reader has familiarity with the concepts and the requirements that are detailed in [MS-RPCE] and [C706]. Click here to view this version of the [MS-EERR] PDF.
[MS-EFSR]: Encrypting File System Remote (EFSRPC) Protocol	Specifies the Encrypting File System Remote (EFSRPC) Protocol, which performs maintenance and management operations on encrypted data that is stored remotely and accessed over a network. Click here to view this version of the [MS-EFSR] PDF.
[MS-EMF]: Enhanced Metafile Format	Specifies the Enhanced Metafile Format (EMF) structure, which can store a picture in device-independent form. Click here to view this version of the [MS-EMF] PDF.

Specification	Description
[MS-EMFPLUS]: Enhanced Metafile Format Plus Extensions	Specifies the Enhanced Metafile Format Plus Extensions, which defines a device-independent structure that encapsulates graphics commands and objects for storage or for sending to devices, such as displays and printers that support the drawing of images, graphics, and text. Click here to view this version of the [MS-EMFPLUS] PDF.
[MS-EMFSPOOL]: Enhanced Metafile Spool Format	Specifies the Enhanced Metafile Spool Format. This structure specifies a metafile format that can store a print job in portable form. Click here to view this version of the [MS-EMFSPOOL] PDF.
[MS-EVEN]: EventLog Remoting Protocol	Specifies the EventLog Remoting Protocol, which exposes the RPC methods for reading events in both live and backup event logs on remote computers. Click here to view this version of the [MS-EVEN] PDF.
[MS-EVEN6]: EventLog Remoting Protocol Version 6.0	Specifies the EventLog Remoting Protocol Version 6.0 protocol, which exposes RPC methods for reading events in both live and backup event logs on remote computers. This protocol was originally made available for Windows Vista. Click here to view this version of the [MS-EVEN6] PDF.
[MS-FASP]: Firewall and Advanced Security Protocol	Specifies the Firewall and Advanced Security Protocol. The protocol manages firewall and advanced security components on remote computers. Click here to view this version of the [MS-FASP] PDF.
[MS-FAX]: Fax Server and Client Remote Protocol	Specifies the Fax Server and Client Remote Protocol. It is an RPC-based, client-server protocol, and is used to send faxes and to manage the fax server and its queues. Click here to view this version of the [MS-FAX] PDF.
[MS-FCIADS]: File Classification Infrastructure Alternate Data Stream (ADS) File Format	Specifies the File Classification Infrastructure Alternate Data Stream (ADS) File Format, which consists of structures for persisting file metadata information into NTFS alternate data streams. Click here to view this version of the [MS-FCIADS] PDF.
[MS-FRS1]: File Replication Service Protocol	Specifies the File Replication Service Protocol, which is a replication protocol that is used to replicate files and folders across one or more members in an Active Directory domain. It works to keep copies of a file system tree up to date on all members of a

Specification	Description
	replication group, while allowing any member of the group to change the contents at any time. Click here to view this version of the [MS-FRS1] PDF.
[MS-FRS2]: Distributed File System Replication Protocol	Specifies the SD Microsoft Distributed File System Replication Protocol, which defines an RPC interface that replicates files between servers and enables the creation of multimaster optimistic file replication systems. Click here to view this version of the [MS-FRS2] PDF.
[MS-FSA]: File System Algorithms	Specifies File System Algorithms in terms of an abstract model for how an object store can be implemented to support the Server Message Block (SMB) Version 1.0 Protocol [MS-SMB] and the Server Message Block (SMB) Version 2.0 Protocol [MS-SMB2]. Click here to view this version of the [MS-FSA] PDF.
[MS-FSCC]: File System Control Codes	Specifies the File System Control Codes that define the network format of native Windows structures that may be used within other protocols. Click here to view this version of the [MS-FSCC] PDF.
[MS-FSRM]: File Server Resource Manager Protocol	Specifies the File Server Resource Manager Protocol, which implements a set of a Distributed Component Object Model (DCOM) interfaces for managing the configuration of directory quotas, file screens, and storage report jobs on a machine. Click here to view this version of the [MS-FSRM] PDF.
[MS-FSRVP]: File Server Remote VSS Protocol	Specifies the File Server Remote VSS Protocol, an RPC-based protocol used for creating shadow copies of file shares on a remote computer, and for facilitating backup applications in performing application-consistent backup and restore of data on SMB2 shares. Click here to view this version of the [MS-FSRVP] PDF.
[MS-FSVCA]: File Set Version Comparison Algorithms	Specifies the File Set Version Comparison Algorithms, which is used by the Enterprise Client Synchronization Protocol to build and serialize a compact representation of version state across a data set consisting of files and directories. Click here to view this version of the [MS-FSVCA] PDF.
[MS-FTPS]: File Transfer Protocol over Secure Sockets Layer (FTPS)	Specifies an extension to the File Transfer Protocol over TLS (FTPS). This extends FTPS with a feature known as Implicit SSL and introduces the AUTH SSL message to allow interoperability with legacy FTP clients.

Specification	Description
	Click here to view this version of the [MS-FTPS] PDF.
[MS-GIPUSB]: Gaming Input Protocol (GIP) Universal Serial Bus (USB) Extension	Specifies the Gaming Input Protocol (GIP) Universal Serial Bus (USB) Extension of the USB 2.0 interface that provides extended semantics for interaction between game controller devices and a host. This protocol includes enumeration of device capabilities, determining of device type and subtype, transfer of gamepad and voice data, and support for an expansion device on the controller. Click here to view this version of the [MS-GIPUSB] PDF.
[MS-GKDI]: Group Key Distribution Protocol	Specifies the Group Key Distribution Protocol, which enables clients to obtain cryptographic keys associated with Active Directory security principals. Click here to view this version of the [MS-GKDI] PDF.
[MS-GPAC]: Group Policy: Audit Configuration Extension	Specifies the Group Policy: Audit Configuration Extension, which provides a mechanism for an administrator to control audit policies on clients. Click here to view this version of the [MS-GPAC] PDF.
[MS-GPCAP]: Group Policy: Central Access Policies Protocol Extension	Specifies the Group Policy: Central Access Policies Extension, which provides the means of configuring central access policies that are applied to Group Policy client computer resources for authorization purposes. Click here to view this version of the [MS-GPCAP] PDF.
[MS-GPDPC]: Group Policy: Deployed Printer Connections Extension	Specifies the Group Policy: Deployed Printer Connections Extension, which supports the use of preconfigured collections of shared printer connections. Click here to view this version of the [MS-GPDPC] PDF.
[MS-GPEF]: Group Policy: Encrypting File System Extension	Specifies the Group Policy: Encrypting File System Extension, which uses the Microsoft Group Policy Protocol to enable remote administrative configuration of the Encrypting File System. Click here to view this version of the [MS-GPEF] PDF.
[MS-GPFAS]: Group Policy: Firewall and Advanced Security Data Structure	Specifies The Group Policy: Firewall and Advanced Security data structure extension, which provides a mechanism for an administrator to control the Firewall and Advanced Security behavior of the client through group policy by using the Group Policy: Registry Extension Encoding protocol [MS-GPREG]. Click here to view this version of the [MS-GPFAS] PDF.

Specification	Description
[MS-GPFR]: Group Policy: Folder Redirection Protocol Extension	Specifies the Group Policy: Folder Redirection Protocol Extension, which provides a mechanism to relocate specific user folders to server disk volumes. The protocol extension describes how file system access requests to a user's folders are automatically redirected to a newly created folder for each user. Click here to view this version of the [MS-GPFR] PDF.
[MS-GPIE]: Group Policy: Internet Explorer Maintenance Extension	Specifies the Group Policy: Internet Explorer Maintenance Extension, which enables administrators to apply custom settings to the Internet Explorer configuration on one or more computers to enforce Internet-related security standards and provide a common browser interface within the organization. Click here to view this version of the [MS-GPIE] PDF.
[MS-GPIPSEC]: Group Policy: IP Security (IPsec) Protocol Extension	Specifies the IP Security (IPSec) Protocol Extension to the Group Policy: Core Protocol. This extension enables administrators to arbitrarily instruct large groups of client machines to configure their local IPsec/IKE components to provide basic IP traffic filtering, IP data integrity, and (optionally) IP data encryption. Click here to view this version of the [MS-GPIPSEC] PDF.
[MS-GPNAP]: Group Policy: Network Access Protection (NAP) Extension	Specifies the Group Policy: Network Access Protection (NAP) Extension, used for controlling access to network resources. This extension enables network administrators to grant or restrict access to network resources based on client computer identity and compliance with corporate governance policy. Click here to view this version of the [MS-GPNAP] PDF.
[MS-GPNRPT]: Group Policy: Name Resolution Policy Table (NRPT) Data Extension	Specifies the Name Resolution Policy Table (NRPT) Group Policy Data Extension, an extension to Group Policy: Registry Extension Encoding [MS-GPREG]. The NRPT Group Policy Data Extension provides a mechanism for an administrator to control any Name Resolution Policy behavior on a client by using group policy-based settings. Click here to view this version of the [MS-GPNRPT] PDF.
[MS-GPOL]: Group Policy: Core Protocol	Specifies the Group Policy: Core Protocol, which enables clients to discover and retrieve policy settings that administrators of a domain create. Click here to view this version of the [MS-GPOL] PDF.

Specification	Description
[MS-GPPREF]: Group Policy: Preferences Extension Data Structure	Specifies the Group Policy: Preferences Extension. This extension to the Group Policy: Core Protocol provides a mechanism to manage and deploy policy preferences. Click here to view this version of the [MS-GPPREF] PDF.
[MS-GPREG]: Group Policy: Registry Extension Encoding	Specifies the Group Policy: Registry Extension Encoding, an extension to the Group Policy: Core Protocol. This mechanism enables an administrator to control any behavior on a client that depends on registry-based settings. Click here to view this version of the [MS-GPREG] PDF.
[MS-GPSB]: Group Policy: Security Protocol Extension	Specifies the Group Policy: Security Protocol Extension, which is an extension to the Group Policy: Core Protocol. This extension enables security policies to be distributed to multiple client systems, so these systems can enact the policies in accordance with the intentions of the administrator. Click here to view this version of the [MS-GPSB] PDF.
[MS-GPSCR]: Group Policy: Scripts Extension Encoding	Specifies the Group Policy: Scripts Extension Encoding, an extension to the Group Policy: Core Protocol that provides a mechanism for an administrator to instruct an arbitrarily large group of clients to execute administrator-specified code at computer startup, computer shutdown, user logon, and user logoff. Click here to view this version of the [MS-GPSCR] PDF.
[MS-GPSI]: Group Policy: Software Installation Protocol Extension	Specifies the Group Policy: Software Installation Protocol Extension, which enables an administrator to install and remove software applications on client computers. Click here to view this version of the [MS-GPSI] PDF.
[MS-GPWL]: Group Policy: Wireless/Wired Protocol Extension	Specifies the Group Policy: Wireless/Wired Protocol Extension, an extension to the Group Policy: Core Protocol that specifies the behaviors of the Wireless/Wired Group Policy administrative-side and client-side plug-in extensions. Click here to view this version of the [MS-GPWL] PDF.
[MS-GSSA]: Generic Security Service Algorithm for Secret Key Transaction Authentication for DNS (GSS-TSIG) Protocol Extension	Specifies the Generic Security Service Algorithm for Secret Key Transaction Authentication for DNS (GSS-TSIG) Protocol Extension, which identifies one possible extension to TSIG based on the Generic Security Service Application Program Interface (GSS-API). Click here to view this version of the [MS-GSSA] PDF.

Specification	Description
[MS-H245]: H.245 Protocol: Microsoft Extensions	Specifies the H.245 Protocol: Microsoft Extensions, which describes Microsoft extensions for the H.323 protocol. Click here to view this version of the [MS-H245] PDF.
[MS-H26XPF]: Real-Time Transport Protocol (RTP/RTCP): H.261 and H.263 Video Streams Extensions	Specifies the Real-Time Transport Protocol (RTP/RTCP): H.261 and H.263 Video Streams Extensions, which are used to transmit and receive H.261 or H.263 video streams in a two-party, peer-to-peer call. Click here to view this version of the [MS-H26XPF] PDF.
[MS-HCEP]: Health Certificate Enrollment Protocol	Specifies the Health Certificate Enrollment Protocol, which enables a network endpoint to obtain digital certificates. Click here to view this version of the [MS-HCEP] PDF.
[MS-HGRP]: HomeGroup Protocol	Specifies the HomeGroup Protocol, which is used to create a trust relationship that facilitates the advertising and publishing of content between machines via a peer-to-peer (P2P) infrastructure. Click here to view this version of the [MS-HGRP] PDF.
[MS-HGSA]: Host Guardian Service: Attestation Protocol	Specifies the Host Guardian Services Attestation (HGSA) protocol, one of two services that comprise the Host Guardian Service. Host Guardian Service is a server role that provides security assurance for Shielded Virtual Machines (VMs) by ensuring that Shielded VMs can be run only on known and trusted fabric hosts that have a legitimate configuration. The other component service, the Key Protection Service, is specified in the [MS-KPS] protocol document. Click here to view this version of the [MS-HGSA] PDF.
[MS-HNDS]: Host Name Data Structure Extension	Specifies the Host Name Data Structure Extension, which defines the allowable host names that may be assigned to a computer. Click here to view this version of the [MS-HNDS] PDF.
[MS-HRL]: Hyper-V Replica Log (HRL) File Format	Specifies the Hyper-V Replica Log (HRL) File Format. Hyper-V Replica log files, required for tracking changes that have been made to the primary server, are created as part of failover replication. They are transported to the recovery server and parsed; updates are then applied to the recovery server. Click here to view this version of the [MS-HRL] PDF.
[MS-HTTP2E]: Hypertext Transfer Protocol Version 2 (HTTP/2) Extension	Specifies a profile of and an extension to the Hypertext Transfer Protocol (HTTP) version 2, which is defined by [RFC7540]. Click here to view this version of the [MS-HTTP2E] PDF.

Specification	Description
[MS-HTTPE]: Hypertext Transfer Protocol (HTTP) Extensions	Specifies the Hypertext Transfer Protocol (HTTP) Extensions, which extend the HyperText Transfer Protocol (HTTP) and deal with internationalization of host names and query strings. Click here to view this version of the [MS-HTTPE] PDF.
[MS-HVRS]: Hyper-V Remote Storage Profile	Specifies information regarding the implementation for hosting Hyper-V virtual machine files on Server Message Block (SMB) Version 3 shares. Click here to view this version of the [MS-HVRS] PDF.
[MS-ICPR]: ICertPassage Remote Protocol	Specifies the ICertPassage Remote Protocol, a subset of the Windows Client Certificate Enrollment Protocol, as specified in [MS-WCCE]. This protocol only enables the client to enroll certificates, whereas [MS-WCCE] provides enrollment and additional functionality. Click here to view this version of the [MS-ICPR] PDF.
[MS-IISS]: Internet Information Services (IIS) ServiceControl Protocol	Specifies the Internet Information Services (IIS) ServiceControl Protocol, a client-to-server protocol that enables remote control of Internet services as a single unit. Click here to view this version of the [MS-IISS] PDF.
[MS-IKEE]: Internet Key Exchange Protocol Extensions	Specifies the Internet Key Exchange (IKE) Protocol Extensions, which describe the extensions specified in [RFC2409]. Click here to view this version of the [MS-IKEE] PDF.
[MS-IMSA]: Internet Information Services (IIS) IMSAdminBaseW Remote Protocol	Specifies the Internet Information Services (IIS) IMSAdminBaseW Remote Protocol, which defines interfaces that provide Unicode-compliant methods for remotely accessing and administering the IIS metabase associated with an application that manages IIS configuration, such as the IIS snap-in for Microsoft Management Console (MMC). Click here to view this version of the [MS-IMSA] PDF.
[MS-IOI]: IManagedObject Interface Protocol	Specifies the IManagedObject Interface Protocol. The IManagedObject interface is a COM interface used by the common language runtime (CLR) to identify managed objects (objects created by the CLR) that are exported for interoperability with the Component Object Model (COM). Click here to view this version of the [MS-IOI] PDF.
[MS-IPAMM]: IP Address Management (IPAM)	Specifies the IP Address Management (IPAM) Management Protocol. This protocol is used to remotely retrieve and manage the

Specification	Description
Management Protocol	data in the IPAM data store. The IPAM data store consists of the data pertaining to address space management, which includes the configuration data available with the DHCP and DNS server instances in the network. Click here to view this version of the [MS-IPAMM] PDF.
[MS-IPAMM2]: IP Address Management (IPAM) Management Protocol Version 2	Specifies the IP Address Management (IPAM) Management Protocol. This protocol is used to remotely retrieve and manage the data in the IPAM data store. The IPAM data store consists of the data pertaining to the address space management, which includes the configuration data available with the DHCP and DNS server instances in the network. Click here to view this version of the [MS-IPAMM2] PDF.
[MS-IPHTTPS]: IP over HTTPS (IP-HTTPS) Tunneling Protocol	Specifies the IP over HTTPS (IP-HTTPS) Tunneling Protocol, a mechanism to transport IPv6 packets on an HTTPS connection. Click here to view this version of the [MS-IPHTTPS] PDF.
[MS-IRDA]: IrDA Object Exchange (OBEX) Protocol Profile	Specifies the IrDA Object Exchange (OBEX) Protocol Profile, which clarifies the implementation details of [IROBEX] where necessary and clarifies which portions of [IROBEX] are not implemented. Click here to view this version of the [MS-IRDA] PDF.
[MS-IRP]: Internet Information Services (IIS) Inetinfo Remote Protocol	Specifies the Internet Information Services (IIS) Inetinfo Remote Protocol, an RPC-based client/server protocol that is used for managing Internet protocol servers such as those hosted by IIS. Click here to view this version of the [MS-IRP] PDF.
[MS-KILE]: Kerberos Protocol Extensions	Specifies the Microsoft implementation of the Kerberos Protocol Extensions, as specified in [RFC4120], by specifying any Windows behaviors that differ from the Kerberos Protocol, in addition to Windows extensions for interactive logon and the inclusion of authorization information expressed as group memberships and related information. Click here to view this version of the [MS-KILE] PDF.
[MS-KKDCP]: Kerberos Key Distribution Center (KDC) Proxy Protocol	Specifies the Kerberos Key Distribution Center (KDC) Proxy Protocol, which provides a mechanism for a client to use a KKDCP server to change passwords and securely obtain Kerberos service tickets from a Kerberos V5 server. Click here to view this version of the [MS-KKDCP] PDF.

Specification	Description
[MS-KPP]: Key Provisioning Protocol	Specifies the Key Provisioning Protocol, which defines a mechanism for a client to register a set of cryptographic keys on a user and device pair. Click here to view this version of the [MS-KPP] PDF.
[MS-KPS]: Key Protection Service Protocol	Specifies the Key Protection Service protocol, one of two services that comprise the Host Guardian Service. Host Guardian Service is a server role that provides security assurance for Shielded Virtual Machines (VMs) by ensuring that Shielded VMs can be run only on known and trusted fabric hosts that have a legitimate configuration. The other component service, the Attestation Service, is specified in the [MS-HGSA] protocol document. Click here to view this version of the [MS-KPS] PDF.
[MS-L2TPIE]: Layer 2 Tunneling Protocol (L2TP) IPsec Extensions	Specifies the Layer 2 Tunneling Protocol (L2TP) IPsec Extensions, which allows IP, IPX, or NetBEUI traffic to be encrypted and then sent over any medium that supports point-to-point (PPP) (Point to Point Protocol [RFC1661]) datagram delivery, such as IP, X.25, Frame Relay, or ATM. Click here to view this version of the [MS-L2TPIE] PDF.
[MS-LLMNRP]: Link Local Multicast Name Resolution (LLMNR) Profile	Specifies the Link Local Multicast Name Resolution (LLMNR) Profile, which describes the differences between this profile and the one defined in [RFC4795]. Click here to view this version of the [MS-LLMNRP] PDF.
[MS-LLTD]: Link Layer Topology Discovery (LLTD) Protocol	Specifies the Link Layer Topology Discovery (LLTD) Protocol, which an application or a higher-layer protocol can use to facilitate discovery of link-layer topology and diagnose various problems associated with a network's signal strength and bandwidth. Click here to view this version of the [MS-LLTD] PDF.
[MS-LREC]: Live Remote Event Capture (LREC) Protocol	Specifies the Live Remote Event Capture (LREC) Protocol, which enables a management station to monitor events on a target system across a network. The protocol supports various monitoring scenarios, such as a ""first line of defense"" for troubleshooting, where the remote system does not support the ability to log events locally. Click here to view this version of the [MS-LREC] PDF.
[MS-LSAD]: Local Security Authority (Domain Policy) Remote Protocol	Specifies the Local Security Authority (Domain Policy) Remote Protocol. This protocol provides an RPC interface used for providing remote management for policy settings related to account objects,

Specification	Description
	secret objects, trusted domain objects (TDOs), and other security-related policy settings. Click here to view this version of the [MS-LSAD] PDF.
[MS-LSAT]: Local Security Authority (Translation Methods) Remote Protocol	Specifies the Local Security Authority (Translation Methods) Remote Protocol, which is implemented in Windows-based products to translate identifiers for security principal between human-readable and machine-readable forms. Click here to view this version of the [MS-LSAT] PDF.
[MS-LWSSP]: Lightweight Web Services Security Profile	Specifies the Lightweight Web Services Security Profile. This profile specifies how to perform lightweight client authentication and security token exchange based on set of security-related Web services protocols. Click here to view this version of the [MS-LWSSP] PDF.
[MS-MAIL]: Remote Mailslot Protocol	Specifies the Remote Mailslot Protocol. This protocol is a simple, nonsecure, and unidirectional interprocess communications (IPC) protocol between a client and server. Click here to view this version of the [MS-MAIL] PDF.
[MS-MCIS]: Content Indexing Services Protocol	Specifies the Content Indexing Services Protocol, which enables a client to communicate with a server hosting an indexing service to issue queries. Click here to view this version of the [MS-MCIS] PDF.
[MS-MDE]: Mobile Device Enrollment Protocol	Specifies the Mobile Device Management Enrollment Protocol, which provides a mechanism for discovering devices and enrolling them into a management system. After enrollment, devices can be managed through the Microsoft Mobile Device Management Protocol [MS-MDM]. Click here to view this version of the [MS-MDE] PDF.
[MS-MDE2]: Mobile Device Enrollment Protocol Version 2	Specifies version 2 of the Mobile Device Enrollment Protocol (MDE), which enables enrolling a device with the DMS through an Enrollment Service (ES). The protocol includes the discovery of the Management Enrollment Service (MES) and enrollment with the ES. Click here to view this version of the [MS-MDE2] PDF.
[MS-MDM]: Mobile Device Management Protocol	Specifies the Mobile Device Management Protocol (MDM), a subset of the Open Mobile Association (OMA) standard protocol, which provides a mechanism for managing devices previously enrolled into a management system through the Microsoft Mobile Device Management Enrollment Protocol [MS-MDE].

Specification	Description
	Click here to view this version of the [MS-MDM] PDF.
[MS-MICE]: Miracast over Infrastructure Connection Establishment Protocol	The Miracast over Infrastructure Connection Establishment Protocol specifies a connection negotiation sequence used to connect, indicate readiness to connect, and disconnect from a Miracast over Infrastructure endpoint. This protocol also specifies the Miracast over Infrastructure Information Element (IE), which helps identify Miracast receivers (sinks) that can support a Miracast session over an infrastructure link (as opposed to a Wi-Fi Direct link). Click here to view this version of the [MS-MICE] PDF.
[MS-MMSP]: Microsoft Media Server (MMS) Protocol	Specifies the Microsoft Media Server (MMS) Protocol, which defines how MMS streams multimedia from Windows Media Services to Windows Media Player, or to another instance of Windows Media Services. MMS uses TCP (Transmission Control Protocol) and UDP (User Datagram Protocol). Click here to view this version of the [MS-MMSP] PDF.
[MS-MNPR]: Microsoft NetMeeting Protocol	Specifies the Microsoft NetMeeting Protocol, which implements a method of application sharing over the T.120 Multipoint Communication Service (MCS) layer, using the S20 MCS Channel. Click here to view this version of the [MS-MNPR] PDF.
[MS-MQBR]: Message Queuing (MSMQ): Binary Reliable Message Routing Algorithm	Specifies the Message Queuing (MSMQ): Binary Reliable Message Routing Algorithm, which is used by MSMQ to communicate across both connected networks and heterogeneous networks. Click here to view this version of the [MS-MQBR] PDF.
[MS-MQCN]: Message Queuing (MSMQ): Directory Service Change Notification Protocol	Specifies the Message Queuing (MSMQ): Directory Service Change Notification Protocol. It defines a mechanism used by the MSMQ Directory Service or a queue manager to notify a queue manager of changes to its owned objects. Click here to view this version of the [MS-MQCN] PDF.
[MS-MQDMPR]: Message Queuing (MSMQ): Common Data Model and Processing Rules	Specifies the Message Queuing (MSMQ): Data Structures, which define an abstract data model and events shared by multiple MSMQ protocols. Click here to view this version of the [MS-MQDMPR] PDF.
[MS-MQDS]: Message Queuing (MSMQ): Directory Service Protocol	Specifies the Message Queuing (MSMQ): Directory Service Protocol, an RPC-based protocol that is used by MSMQ clients and Message Queuing servers to remotely access and maintain MSMQ directory objects.

Specification	Description
	Click here to view this version of the [MS-MQDS] PDF.
[MS-MQDSSM]: Message Queuing (MSMQ): Directory Service Schema Mapping	Specifies the Message Queuing (MSMQ): Data Structures that are used by any protocol that manipulates the subset of the abstract data elements and data element attributes defined in [MS-MQDMPR] section 3.1. Click here to view this version of the [MS-MQDSSM] PDF.
[MS-MQMP]: Message Queuing (MSMQ): Queue Manager Client Protocol	Specifies the Message Queuing (MSMQ): Queue Manager Client Protocol, which enables communication between message queuing client applications and an MSMQ Queue Manager. Click here to view this version of the [MS-MQMP] PDF.
[MS-MQMQ]: Message Queuing (MSMQ): Data Structures	Specifies Message Queuing (MSMQ): Data Structures, which contains common definitions and data structures that are used in the Microsoft Message Queuing protocols. Click here to view this version of the [MS-MQMQ] PDF.
[MS-MQMR]: Message Queuing (MSMQ): Queue Manager Management Protocol	Specifies the Message Queuing (MSMQ): Queue Manager Management Protocol that is used for management operations on the MSMQ server, including monitoring the MSMQ installation and the queues. Click here to view this version of the [MS-MQMR] PDF.
[MS-MQQB]: Message Queuing (MSMQ): Message Queuing Binary Protocol	Specifies the Message Queuing (MSMQ): Binary Reliable Messaging Protocol, which defines a mechanism for reliably transferring messages between two message queues located on two different hosts. Click here to view this version of the [MS-MQQB] PDF.
[MS-MQQP]: Message Queuing (MSMQ): Queue Manager to Queue Manager Protocol	Specifies the Message Queuing (MSMQ): Queue Manager to Queue Manager Protocol, an RPC-based protocol used by the queue manager and runtime library to read and purge messages from a remote queue. Click here to view this version of the [MS-MQQP] PDF.
[MS-MQRR]: Message Queuing (MSMQ): Queue Manager Remote Read Protocol	Specifies the Message Queuing (MSMQ): Queue Manager Remote Read Protocol, an RPC-based protocol that is used by MSMQ clients to read or reject a message from a queue, move a message between queues, and purge messages from a queue. Click here to view this version of the [MS-MQRR] PDF.
[MS-MQSD]: Message Queuing (MSMQ):	Specifies the Message Queuing (MSMQ): Directory Service Discovery Protocol, which is used by MSMQ clients to discover an

Specification	Description
Directory Service Discovery Protocol	accessible executing instance of an MSMQ Directory Service server. Click here to view this version of the [MS-MQSD] PDF.
[MS-MSB]: Media Stream Broadcast (MSB) Protocol	Specifies the Media Stream Broadcast (MSB) Protocol, which enables distribution of Advanced Systems Format (ASF) packets over a network for which Internet Protocol (IP) multicasting is enabled. Click here to view this version of the [MS-MSB] PDF.
[MS-MSBD]: Media Stream Broadcast Distribution (MSBD) Protocol	Specifies the Media Stream Broadcast Distribution (MSBD) Protocol, which describes how to transfer an audio-visual content stream from a server to a single client. Click here to view this version of the [MS-MSBD] PDF.
[MS-MSRP]: Messenger Service Remote Protocol	Specifies the Messenger Service Remote Protocol, a set of RPC interfaces that instructs a server to display short text messages to a console user, to deliver messages to a local or remote server for display to a console user, and to manage the names for which the server receives messages. Click here to view this version of the [MS-MSRP] PDF.
[MS-MWBE]: Microsoft Web Browser Federated Sign-On Protocol Extensions	Specifies the Microsoft Web Browser Federated Sign-On Protocol Extensions. This extension enables Web browser requestors that do not support scripting (to create POST messages) and enables passing security identifiers (SIDs) in Security Assertion Markup Language (SAML) V1.1 assertions. It is assumed that the reader is familiar with the terms, concepts, and protocols that are defined in [MS-MWBF]. Click here to view this version of the [MS-MWBE] PDF.
[MS-MWBF]: Microsoft Web Browser Federated Sign-On Protocol	Specifies the Microsoft Web Browser Federated Sign-On Protocol, which is primarily a restriction of the protocol that is specified in [WSFederation1.2] section 13. The restrictions are designed to enable greater interoperability by reducing the number of variations that must be implemented. This protocol also specifies minor additions to [WSFederation1.2] section 13 to handle common scenarios. Click here to view this version of the [MS-MWBF] PDF.
[MS-N2HT]: Negotiate and Nego2 HTTP Authentication Protocol	Specifies the Negotiate and Nego2 HTTP Authentication Protocol, which describes support for SPNEGO authentication as specified in [RFC4559]. The tokens are transmitted using base64-encoding. This protocol calls out the differences in the Microsoft implementation from what is specified in [RFC4559], where applicable.

Specification	Description
	Click here to view this version of the [MS-N2HT] PDF.
[MS-NBTE]: NetBIOS over TCP (NBT) Extensions	Specifies the NetBIOS over TCP (NBT) Extensions, as specified in [RFC1001] and [RFC1002]. These extensions modify the syntax of allowable NetBIOS names and the behavior of timers, and add support for multihomed hosts. Click here to view this version of the [MS-NBTE] PDF.
[MS-NCNBI]: Network Controller Northbound Interface	Specifies the Network Controller Protocol, which is used by tenants and network administrators to control data center networking. Common tasks that would use these APIs include designing and monitoring a virtual network in a data center. Click here to view this version of the [MS-NCNBI] PDF.
[MS-NCT]: Network Cost Transfer Protocol	Enables an 802.11 wireless access point (AP) to inform a wireless client of the network cost and hints about the AP type. Click here to view this version of the [MS-NCT] PDF.
[MS-NEGOEX]: SPNEGO Extended Negotiation (NEGOEX) Security Mechanism	Specifies the SPNEGO Extended Negotiation (NEGOEX) Security Mechanism that enhances the capabilities of SPNEGO [RFC4178] by providing a security mechanism which can be negotiated by SPNEGO. When the NEGOEX security mechanism is selected by SPNEGO, NEGOEX provides a method allowing selection of a common authentication protocol based on meta-data such as trust configurations. Click here to view this version of the [MS-NEGOEX] PDF.
[MS-NETTR]: .NET Tracing Protocol	Specifies the .NET Tracing Protocol, which provides a method for correlating call traces in a .NET remoting application. Click here to view this version of the [MS-NETTR] PDF.
[MS-NFPB]: Near Field Proximity: Bidirectional Services Protocol	Specifies the Near Field Proximity: Bidirectional Services Protocol, which provides a way for devices to discover services and versions from one device to another. The protocol uses the ""Proximity Publication Subscription"" transport to exchange messages between peers. Click here to view this version of the [MS-NFPB] PDF.
[MS-NFPS]: Near Field Proximity: Sharing Protocol	Specifies the Near Field Proximity: Sharing Protocol, which provides a way for devices to share files over an already established single-purpose channel. A client can use this protocol to send a set of files packaged in an Open Packaging Convention (OPC) file and encrypted over the channel.

Specification	Description
	Click here to view this version of the [MS-NFPS] PDF.
[MS-NKPU]: Network Key Protector Unlock Protocol	Specifies the Network Key Protector Unlock Protocol, which enables a client to send an encrypted package of key material along with a session key to a remote server and to receive the decrypted key material protected by the session key. Click here to view this version of the [MS-NKPU] PDF.
[MS-NLMP]: NT LAN Manager (NTLM) Authentication Protocol	Specifies the NT LAN Manager (NTLM) Authentication Protocol, used in Windows for authentication between clients and servers. NTLM is used by application protocols to authenticate remote users and, optionally, to provide session security when requested by the application. Click here to view this version of the [MS-NLMP] PDF.
[MS-NMFMB]: .NET Message Framing MSMQ Binding Protocol	Specifies the .NET Message Framing MSMQ Binding Protocol, which defines how the mechanism described in [MC-NMF] for framing messages over any transport protocol can be applied over Message Queue (MSMQ). This protocol specification also defines how to indicate the use of .NET Message Framing over MSMQ as a SOAP transport in Web Services Description Language (WSDL). Click here to view this version of the [MS-NMFMB] PDF.
[MS-NMFTB]: .NET Message Framing TCP Binding Protocol	Specifies how the .NET Message Framing Protocol [MC-NMF] is bound to a TCP connection, including the initiation of the stream by using the net.tcp URI scheme and the application of .NET Message Framing over TCP as a SOAP transport in WSDL. Click here to view this version of the [MS-NMFTB] PDF.
[MS-NNS]: .NET NegotiateStream Protocol	Specifies the .NET NegotiateStream Protocol, which provides mutually authenticated and confidential communication over a TCP connection. It uses the Simple and Protected GSS-API Negotiation mechanism (SPNEGO) for security services (authentication, key derivation, and data encryption and decryption). Click here to view this version of the [MS-NNS] PDF.
[MS-NNTP]: NT LAN Manager (NTLM) Authentication: Network News Transfer Protocol (NNTP) Extension	Specifies the NT LAN Manager (NTLM) Authentication: Network News Transfer Protocol (NNTP) Extension, which defines the use of NTLM authentication by NNTP to facilitate client authentication to a Windows-based NNTP server. Click here to view this version of the [MS-NNTP] PDF.
[MS-NRBF]: .NET Remoting: Binary Format	Specifies the .NET Remoting: Binary Format Data Structure protocol, which defines a set of structures for representing object graph or

Specification	Description
Data Structure	method invocation information as an octet stream. Click here to view this version of the [MS-NRBF] PDF.
[MS-NRLS]: .NET Remoting: Lifetime Services Extension	Specifies the .NET Remoting: Lifetime Services Extension, which adds lifetime and remote activation capabilities to the .NET Remoting Core Protocol (specified in [MS-NRTP]). Click here to view this version of the [MS-NRLS] PDF.
[MS-NRPC]: Netlogon Remote Protocol	Specifies the Netlogon Remote Protocol, an RPC interface that is used for user and machine authentication on domain-based networks; to replicate the user account database for operating systems earlier than Windows 2000 backup domain controllers; to discover, manage, and maintain domain relationships of domain members and domain controllers across domains. Click here to view this version of the [MS-NRPC] PDF.
[MS-NRTP]: .NET Remoting: Core Protocol	Specifies the .NET Remoting: Core Protocol, a mechanism by which a calling program can invoke a method in a different address space over the network. Arguments are passed along as part of the invocation message, and return values are sent in the response. Click here to view this version of the [MS-NRTP] PDF.
[MS-NSPI]: Name Service Provider Interface (NSPI) Protocol	Specifies the Name Service Provider Interface (NSPI) Protocol, which provides messaging clients with a way to access and manipulate addressing data stored by a server. This protocol consists of an abstract data model and a single RPC call interface to manipulate data in that model. Click here to view this version of the [MS-NSPI] PDF.
[MS-NTHT]: NTLM Over HTTP Protocol	Specifies the NTLM Over HTTP Protocol, which is used to authenticate a Web client to a Web server. This protocol authentication variant works only with NTLM; the Kerberos protocol is not supported. Click here to view this version of the [MS-NTHT] PDF.
[MS-NVGREE]: Network Virtualization using Generic Routing Encapsulation (NVGRE) Extensions	Specifies the Network Virtualization using Generic Routing Encapsulation (NVGRE) Extensions protocol, which adds additional support to the NVGRE protocol by defining two new extension/control messages. One message initiates traffic redirection to a new network virtualization endpoint (NVE) when a VM is moved to a new NVE during a data center migration event. The other new message pushes out a policy refresh via an NVE in response to the moved VM.

Specification	Description
	Click here to view this version of the [MS-NVGREE] PDF.
[MS-OAPX]: OAuth 2.0 Protocol Extensions	Specifies the OAuth 2.0 Protocol Extensions, which are used to extend the OAuth 2.0 Authorization Framework. These extensions enable authorization features such as resource specification, request identifiers, and login hints. Click here to view this version of the [MS-OAPX] PDF.
[MS-OAPXBC]: OAuth 2.0 Protocol Extensions for Broker Clients	Specifies the OAuth 2.0 Protocol Extensions for Broker Clients, extensions to [RFC6749] (The OAuth 2.0 Authorization Framework) that allow a broker client to obtain access tokens on behalf of calling clients. Click here to view this version of the [MS-OAPXBC] PDF.
[MS-OAUT]: OLE Automation Protocol	Specifies the OLE Automation Protocol, which uses DCOM as its transport layer and provides support for an additional set of types as well as for a late-bound calling mechanism. Click here to view this version of the [MS-OAUT] PDF.
[MS-OCSP]: Online Certificate Status Protocol (OCSP) Extensions	Specifies the Online Certificate Status Protocol (OCSP) Extensions, which defines the data that needs to be exchanged between an application that checks the status of a certificate and the responder that provides the status. Click here to view this version of the [MS-OCSP] PDF.
[MS-OCSPA]: Microsoft OCSP Administration Protocol	Specifies the Microsoft OCSP Administration Protocol, which consists of a set of distributed component object model (DCOM) interfaces that allows administrative tools to configure the properties of the Online Responder. Click here to view this version of the [MS-OCSPA] PDF.
[MS-ODATA]: Open Data Protocol (OData)	Specifies the Open Data (OData) Protocol. This protocol enables applications to expose data, by using common Web technologies, and by means of a data service that can be consumed by clients within corporate networks and across the Internet. Click here to view this version of the [MS-ODATA] PDF.
[MS-OIDCE]: OpenID Connect 1.0 Protocol Extensions	Specifies the OpenID Connect 1.0 Protocol Extensions. These extensions define additional claims to carry information about the end user, including the user principal name, a locally unique identifier, a time for password expiration, and a URL for password change. These extensions also define additional provider metadata that enable the discovery of the issuer of access tokens and give additional information about provider capabilities.

Specification	Description
	Click here to view this version of the [MS-OIDCE] PDF.
[MS-OLEDS]: Object Linking and Embedding (OLE) Data Structures	Specifies the Object Linking and Embedding (OLE) Data Structures. These structures enable applications to create documents that contain linked or embedded objects. Click here to view this version of the [MS-OLEDS] PDF.
[MS-OLEPS]: Object Linking and Embedding (OLE) Property Set Data Structures	Specifies the Object Linking and Embedding (OLE): Property Set Data Structures. These structures enable applications to write metadata in a manner that is discoverable to other software. Click here to view this version of the [MS-OLEPS] PDF.
[MS-OTPCE]: One-Time Password Certificate Enrollment Protocol	Specifies the One-Time Password Certificate Enrollment Protocol, which enhances network security in remote access connections by utilizing different components, such as the one-time password (OTP) authentication mechanism as well as a short-lived smart card logon certificate. Click here to view this version of the [MS-OTPCE] PDF.
[MS-PAC]: Privilege Attribute Certificate Data Structure	Specifies the Privilege Attribute Certificate Data Structure, which is used to encode authorization information. The Privilege Attribute Certificate also contains memberships, additional credential information, profile and policy information, and supporting security metadata. Click here to view this version of the [MS-PAC] PDF.
[MS-PAN]: Print System Asynchronous Notification Protocol	Specifies the [MS-PAN]: Print System Asynchronous Notification Protocol, an asynchronous protocol that clients use to receive print status notifications from a print server and send server-requested responses to those notifications back to the server. It is based on the Remote Procedure Call (RPC) protocol, as specified in [C706]. Click here to view this version of the [MS-PAN] PDF.
[MS-PAR]: Print System Asynchronous Remote Protocol	Specifies the Print System Asynchronous Remote Protocol, which defines the communication of print job processing and print system management information between a print client and a print server. Click here to view this version of the [MS-PAR] PDF.
[MS-PASS]: Passport Server Side Include (SSI) Version 1.4 Protocol	Specifies the Passport Server Side Include (SSI) Version 1.4 Protocol, which describes how messages are encapsulated on the wire. Click here to view this version of the [MS-PASS] PDF.

Specification	Description
[MS-PBSD]: Publication Services Data Structure	Specifies the Publication Services Data Structure. This structure describes the data that computers use to describe themselves and the resources they offer as Web services over IP-based networks. Click here to view this version of the [MS-PBSD] PDF.
[MS-PCCRC]: Peer Content Caching and Retrieval: Content Identification	Specifies Peer Content Caching and Retrieval: Content Identification, the content information format used by the Windows Branch Caching Framework to uniquely identify content for discovery and retrieval purposes. Click here to view this version of the [MS-PCCRC] PDF.
[MS-PCCRD]: Peer Content Caching and Retrieval: Discovery Protocol	Specifies the Peer Content Caching and Retrieval Discovery Protocol, which is based on the Web Services Dynamic Discovery (WS-Discovery) protocol. It is a content caching and retrieval framework based on a peer-to-peer discovery and distribution model. Click here to view this version of the [MS-PCCRD] PDF.
[MS-PCCRR]: Peer Content Caching and Retrieval: Retrieval Protocol	Specifies the Peer Content Caching and Retrieval: Retrieval Protocol. This protocol defines two message exchanges, one for querying the server for the availability of certain content, and the other for retrieving content from a server. Click here to view this version of the [MS-PCCRR] PDF.
[MS-PCCRTP]: Peer Content Caching and Retrieval: Hypertext Transfer Protocol (HTTP) Extensions	Specifies the Peer Content Caching and Retrieval: Hypertext Transfer Protocol (HTTP) Extensions, which implements a new type of content encoding, PeerDist, that can be used in HTTP/1.1. In particular, it specifies the mechanism used by an HTTP/1.1 client and an HTTP/1.1 server to communicate with each other using the PeerDist content encoding. Click here to view this version of the [MS-PCCRTP] PDF.
[MS-PCHC]: Peer Content Caching and Retrieval: Hosted Cache Protocol	Specifies the Peer Content Caching and Retrieval: Hosted Cache Protocol, used by clients to offer metadata to a hosted cache server. Click here to view this version of the [MS-PCHC] PDF.
[MS-PCQ]: Performance Counter Query Protocol	Specifies the Performance Counter Query Protocol, which is used for browsing performance counters and retrieving performance counter values from a server. Click here to view this version of the [MS-PCQ] PDF.
[MS-PEAP]: Protected Extensible Authentication	Specifies the Protected Extensible Authentication Protocol (PEAP), which adds security services to the Extensible Authentication

Specification	Description
Protocol (PEAP)	Protocol methods. Click here to view this version of the [MS-PEAP] PDF.
[MS-PKAP]: Public Key Authentication Protocol	Specifies the Public Key Authentication Protocol, which provides a method for HTTP clients to prove possession of a private key to a web server without having to rely on client Transport Layer Security (TLS) support from the underlying platform. Click here to view this version of the [MS-PKAP] PDF.
[MS-PKCA]: Public Key Cryptography for Initial Authentication (PKINIT) in Kerberos Protocol	Specifies the Public Key Cryptography for Initial Authentication (PKINIT) in Kerberos Protocol. This protocol enables the use of public key cryptography in the initial authentication exchange of the Kerberos Protocol (PKINIT) and specifies the Windows implementation of PKINIT where it differs from [RFC4556]. Click here to view this version of the [MS-PKCA] PDF.
[MS-PLA]: Performance Logs and Alerts Protocol	Specifies the Performance Logs and Alerts Protocol, which provides a set of DCOM interfaces to control data collection on a remote system. The control includes starting, stopping, scheduling, and configuration of data collector objects, and the creation of alerts. Click here to view this version of the [MS-PLA] PDF.
[MS-PNRP]: Peer Name Resolution Protocol (PNRP) Version 4.0	Specifies the Peer Name Resolution Protocol (PNRP) Version 4.0, which is used to resolve a name to a set of information, such as IP addresses; to maintain a cloud of peer nodes; to maintain a distributed cache of endpoint information; and to transfer requests for Peer Name resolutions between nodes. Click here to view this version of the [MS-PNRP] PDF.
[MS-POP3]: NT LAN Manager (NTLM) Authentication: Post Office Protocol - Version 3 (POP3) Extension	Specifies the NT LAN Manager (NTLM) Authentication: Post Office Protocol - Version 3 (POP3) Extension, which describes the use of NTLM Authentication (see [MS-NLMP]) by the Post Office Protocol 3 (POP3) to facilitate client authentication to a Windows POP3 server. POP3 specifies a protocol for the inquiry and retrieval of electronic mail. Click here to view this version of the [MS-POP3] PDF.
[MS-PPGRH]: Peer-to-Peer Graphing Protocol	Specifies the Peer-to-Peer Graphing Protocol, a peer-to-peer protocol for establishing and maintaining a connected set of nodes (referred to as a graph), and replicating data among the nodes. Click here to view this version of the [MS-PPGRH] PDF.

Specification	Description
[MS-PPPI]: PPP Over IrDA Dialup Protocol	Specifies the PPP Over IrDA Dialup Protocol, which enables the scenario in which a computer with infrared capabilities obtains network access by using a modem via the infrared link. Click here to view this version of the [MS-PPPI] PDF.
[MS-PPSEC]: Peer-to-Peer Grouping Security Protocol	Specifies the Peer-to-Peer Grouping Security Protocol (P2P Grouping), which layers on top of the Peer-to-Peer Graphing Protocol [MS-PPGRH] and adds security and discovery services. Click here to view this version of the [MS-PPSEC] PDF.
[MS-PROPSTORE]: Property Store Binary File Format	Specifies the Property Store Binary File Format. This file format is a persistence format for a set of properties. Implementers can use this file format to store a set of properties in a file or within another structure. Click here to view this version of the [MS-PROPSTORE] PDF.
[MS-PSDP]: Proximity Service Discovery Protocol	Specifies the Proximity Service Discovery Protocol, which conveys service discovery information, such as service advertisements, as part of Beacon frames, as specified in [IEEE802.11-2007]. Click here to view this version of the [MS-PSDP] PDF.
[MS-PSRDP]: PowerShell Remote Debugging Protocol	Specifies the PowerShell Remote Debugging Protocol (PSRDP).This protocol extends the existing PowerShell Remoting Protocol (PSRP) specified in [MS-PSRP] to support debugging over a remote session. Click here to view this version of the [MS-PSRDP] PDF.
[MS-PSRP]: PowerShell Remoting Protocol	Specifies the Windows PowerShell Remoting Protocol, which encodes messages prior to sending them over the Web Services Management Protocol Extensions for the Windows Vista [MS-WSMV] layer. Click here to view this version of the [MS-PSRP] PDF.
[MS-PTPT]: Point-to-Point Tunneling Protocol (PPTP) Profile	Specifies the Point-to-Point Tunneling Protocol, which allows the Point-to-Point Protocol (PPP) [RFC1661] to be tunneled through an IP network. Click here to view this version of the [MS-PTPT] PDF.
[MS-QDP]: Quality Windows Audio/Video Experience (qWave): Wireless Diagnostics Protocol	Specifies the Quality Windows Audio/Video Experience (qWave): Wireless Diagnostics Protocol. This protocol is used to obtain information from a host or a device about its wireless characteristics, which can facilitate the diagnosis of wireless network issues.

Specification	Description
	Click here to view this version of the [MS-QDP] PDF.
[MS-QLPB]: Quality Windows Audio/Video Experience (qWave): Layer 3 Probing Protocol	Specifies the Quality Windows Audio/Video Experience (qWave): Layer 3 Probing (L3P) (qWave) Protocol, which operates over TCP/IP and UDP/IP. qWave enables applications to evaluate link bandwidth and quality by analyzing timestamps of probe packets transmitted between two devices. Click here to view this version of the [MS-QLPB] PDF.
[MS-RA]: Remote Assistance Protocol	Specifies the Remote Assistance Protocol, which is used after a remote assistance connection is established between two computers. Click here to view this version of the [MS-RA] PDF.
[MS-RAA]: Remote Authorization API Protocol	Specifies the Remote Authorization API Protocol, which is used to perform ""what-if"" authorization queries on remote computers. It allows applications to simulate an access control decision that would be made when a principal attempts to access a remote resource protected with an authorization policy. Click here to view this version of the [MS-RAA] PDF.
[MS-RAI]: Remote Assistance Initiation Protocol	Specifies the Remote Assistance Initiation Protocol, which enables an authorized expert to start Remote Assistance (RA) on a remote novice computer to retrieve data that is required to make a Remote Assistance connection from the expert's computer to the novice's computer. Click here to view this version of the [MS-RAI] PDF.
[MS-RAIOP]: Remote Assistance Initiation over PNRP Protocol	Specifies the Remote Assistance Initiation over PNRP Protocol, which is used to establish a Remote Assistance connection between two computers. Click here to view this version of the [MS-RAIOP] PDF.
[MS-RAIW]: Remote Administrative Interface: WINS	Specifies the Remote Administrative Interface: WINS protocol, which enables local or remote administration of the Windows Internet Name Service (WINS) within the Microsoft Management Console (MMC) WINS snap-in and the NetSh command line (WINS context). Click here to view this version of the [MS-RAIW] PDF.
[MS-RAP]: Remote Administration Protocol	Specifies the Microsoft Remote Administration Protocol (RAP), which Microsoft LAN Manager uses to perform remote administrative functions and is included in the Microsoft Windows operating system for compatibility reasons.

Specification	Description
	Click here to view this version of the [MS-RAP] PDF.
[MS-RASA]: Remote Access Server Advertisement (RASADV) Protocol	Specifies the Remote Access Server Advertisement (RASADV) Protocol, by which Remote Access Service (RAS) Servers advertise their presence within a local network, enabling network administrators to detect nonmalicious configuration and deployment of gateways providing external access to their network. Click here to view this version of the [MS-RASA] PDF.
[MS-RCMP]: Remote Certificate Mapping Protocol	Specifies the Remote Certificate Mapping Protocol, which enables servers to use a directory, database, or other technology to map the user's X.509 certificate to a security principal. Click here to view this version of the [MS-RCMP] PDF.
[MS-RDC]: Remote Differential Compression Algorithm	Specifies the Remote Differential Compression Algorithm protocol, which enables efficient synchronization of files with a remote source by using compression techniques to minimize the amount of data sent between a client and server. Click here to view this version of the [MS-RDC] PDF.
[MS-RDPADRV]: Remote Desktop Protocol: Audio Level and Drive Letter Persistence Virtual Channel Extension	Specifies the Remote Desktop Protocol: Audio Level and Drive Letter Persistence Virtual Channel Extension, which allows an RDP (remote desktop connection) client device to mimic a Windows client PC session with respect to audio levels and drive letters. Click here to view this version of the [MS-RDPADRV] PDF.
[MS-RDPBCGR]: Remote Desktop Protocol: Basic Connectivity and Graphics Remoting	Specifies the Remote Desktop Protocol: Basic Connectivity and Graphics Remoting, designed to facilitate user interaction with a remote computer system by transferring graphics display information from the remote computer to the user and transporting input from the user to the remote computer, where it may be injected locally. Click here to view this version of the [MS-RDPBCGR] PDF.
[MS-RDPCR2]: Remote Desktop Protocol: Composited Remoting V2	Specifies the Remote Desktop Protocol: Composited Remoting V2, which displays the contents of the Windows-based desktop running on one machine on a second machine connected to the first via a network. Click here to view this version of the [MS-RDPCR2] PDF.
[MS-RDPEA]: Remote Desktop Protocol: Audio Output Virtual Channel Extension	Specifies the Remote Desktop Protocol: Audio Output Virtual Channel Extension, which transfers audio data from the server to the client.

Specification	Description
	Click here to view this version of the [MS-RDPEA] PDF.
[MS-RDPEAI]: Remote Desktop Protocol: Audio Input Redirection Virtual Channel Extension	Specifies the Remote Desktop Protocol: Audio Input Redirection Virtual Channel Extension, which transfers audio data from a client to a server. Click here to view this version of the [MS-RDPEAI] PDF.
[MS-RDPEAR]: Remote Desktop Protocol Authentication Redirection Virtual Channel	Performs authentication over a Remote Desktop connection. By establishing a virtual channel between the source and the target devices, it can relay authentication requests received by the target device to the source device. Click here to view this version of the [MS-RDPEAR] PDF.
[MS-RDPECAM]: Remote Desktop Protocol: Video Capture Virtual Channel Extension	The Remote Desktop Protocol: Video Capture Virtual Channel Extension adds remoting of video capture devices, such as webcams, to the Basic Connectivity and Graphics Remoting Protocol. Click here to view this version of the [MS-RDPECAM] PDF.
[MS-RDPECI]: Remote Desktop Protocol: Core Input Virtual Channel Extension	The Remote Desktop Protocol: Core Input Virtual Channel Extension enables remoting of keyboard and mouse pointer input over the UDP transport. This is an extension to the Basic Connectivity and Graphics Remoting Protocol (2785). Click here to view this version of the [MS-RDPECI] PDF.
[MS-RDPECLIP]: Remote Desktop Protocol: Clipboard Virtual Channel Extension	Specifies the Remote Desktop Protocol: Clipboard Virtual Channel Extension, which enables users to seamlessly transfer data via the system clipboard between applications that are running on different computers. Click here to view this version of the [MS-RDPECLIP] PDF.
[MS-RDPEDC]: Remote Desktop Protocol: Desktop Composition Virtual Channel Extension	Specifies the Remote Desktop Protocol: Desktop Composition Virtual Channel Extension, which enables a remote display client to replicate the functionality of the Desktop Window Manager (DWM) across a network boundary. Click here to view this version of the [MS-RDPEDC] PDF.
[MS-RDPEDISP]: Remote Desktop Protocol: Display Update Virtual Channel Extension	Specifies the Remote Desktop Protocol: Display Control Virtual Channel Extension to the Remote Desktop Protocol: Basic Connectivity and Graphics Remoting, as specified in [MS-RDPBCGR]. This control protocol is used to request display configuration changes in a remote session. Click here to view this version of the [MS-RDPEDISP] PDF.

Specification	Description
[MS-RDPEDYC]: Remote Desktop Protocol: Dynamic Channel Virtual Channel Extension	Specifies the Remote Desktop Protocol: Dynamic Channel Virtual Channel Extension, which supports features such as classes of priority (that may be used to implement bandwidth allocation) and individually connected endpoints using dynamic virtual channel (DVC) listeners. Click here to view this version of the [MS-RDPEDYC] PDF.
[MS-RDPEECO]: Remote Desktop Protocol: Virtual Channel Echo Extension	Specifies the Remote Desktop Protocol: Virtual Channel Echo Extension. This extension is used as a ping and echo mechanism to determine various network characteristics that are significant for RDP. Click here to view this version of the [MS-RDPEECO] PDF.
[MS-RDPEFS]: Remote Desktop Protocol: File System Virtual Channel Extension	Specifies the Remote Desktop Protocol: File System Virtual Channel Extension, which runs over a static virtual channel with the name RDPDR. Click here to view this version of the [MS-RDPEFS] PDF.
[MS-RDPEGDI]: Remote Desktop Protocol: Graphics Device Interface (GDI) Acceleration Extensions	Specifies the Remote Desktop Protocol: Graphics Device Interface (GDI) Acceleration Extensions, which reduces the bandwidth associated with graphics remoting by encoding the drawing operations that produce an image instead of encoding the actual image. Click here to view this version of the [MS-RDPEGDI] PDF.
[MS-RDPEGFX]: Remote Desktop Protocol: Graphics Pipeline Extension	Specifies the Remote Desktop Protocol: Graphics Pipeline Extension, a graphics protocol that is used to encode graphics display data generated in a remote terminal server session so that the data can be sent from the server and received, decoded, and rendered by a compatible client. The net effect is that a desktop or an application running on a remote terminal server appears as if it is running locally. Click here to view this version of the [MS-RDPEGFX] PDF.
[MS-RDPEGT]: Remote Desktop Protocol: Geometry Tracking Virtual Channel Protocol Extension	Specifies the Remote Desktop Protocol: Geometry Tracking Virtual Channel Extension, which extends the Remote Desktop Protocol: Basic Connectivity and Graphics Remoting. This protocol facilitates graphics rendering between a desktop host and a remote desktop client in a way that the client does not need to know the origin of the graphics. Click here to view this version of the [MS-RDPEGT] PDF.
[MS-RDPEI]: Remote Desktop Protocol: Input	Specifies the Remote Desktop Protocol: Input Virtual Channel Extension, which is used to remote multitouch input frames from a

Specification	Description
Virtual Channel Extension	terminal server client to a terminal server. Multitouch input frames are generated at the client, encoded, and sent to the server. Thereafter, these frames are received and decoded by the server and injected into the session associated with the remote user. Click here to view this version of the [MS-RDPEI] PDF.
[MS-RDPEL]: Remote Desktop Protocol: Location Virtual Channel Extension	Specifies the Remote Desktop Protocol: Location Channel Extension, which adds the ability to redirect the client's location (latitude, longitude and altitude) to a server so that location-based services running in a user session can provide a more contextualized experience where possible. Click here to view this version of the [MS-RDPEL] PDF.
[MS-RDPELE]: Remote Desktop Protocol: Licensing Extension	Specifies the Remote Desktop Protocol: Licensing Extension, which expands on the licensing protocol sequence specified in [MS-RDPBCGR] to address scenarios requiring the exchange of licensing information between the client and server. Click here to view this version of the [MS-RDPELE] PDF.
[MS-RDPEMC]: Remote Desktop Protocol: Multiparty Virtual Channel Extension	Specifies the Remote Desktop Protocol: Multiparty Virtual Channel Extension, which describes the messages that are exchanged between a remote desktop host and the participants with whom it is engaging in multiparty application sharing. Click here to view this version of the [MS-RDPEMC] PDF.
[MS-RDPEMSC]: Remote Desktop Protocol: Mouse Cursor Virtual Channel Extension	The Remote Desktop Protocol: Mouse Cursor Virtual Channel Extension enables remoting of mouse cursor bitmap over the UDP transport. This is an extension to the Basic Connectivity and Graphics Remoting Protocol (2785). Click here to view this version of the [MS-RDPEMSC] PDF.
[MS-RDPEMT]: Remote Desktop Protocol: Multitransport Extension	This document specifies the Remote Desktop Protocol: Multitransport Extension, which is used to create multiple data-transport connections between an RDP client and an RDP server. Click here to view this version of the [MS-RDPEMT] PDF.
[MS-RDPEPC]: Remote Desktop Protocol: Print Virtual Channel Extension	Specifies the Desktop Protocol: Print Virtual Channel Extensions, which specifies the communication used to enable the redirection of printers between a terminal client and a terminal server. Click here to view this version of the [MS-RDPEPC] PDF.
[MS-RDPEPNP]: Remote Desktop Protocol: Plug and	Specifies the Remote Desktop Protocol: Plug and Play Devices Virtual Channel Extension, which is used to redirect Plug and Play devices from a terminal client to the terminal server.

Specification	Description
Play Devices Virtual Channel Extension	Click here to view this version of the [MS-RDPEPNP] PDF.
[MS-RDPEPS]: Remote Desktop Protocol: Session Selection Extension	Specifies the Remote Desktop Protocol: Session Selection Extension, which expands upon the original connectivity options specified in [MS-RDPBCGR] to address a wide range of new scenarios. Click here to view this version of the [MS-RDPEPS] PDF.
[MS-RDPERP]: Remote Desktop Protocol: Remote Programs Virtual Channel Extension	Specifies the Remote Desktop Protocol: Remote Programs Virtual Channel Extension, an RDP feature that presents a remote application (running remotely on a RAIL server) as a local user application (running on the RAIL client machine). Click here to view this version of the [MS-RDPERP] PDF.
[MS-RDPESC]: Remote Desktop Protocol: Smart Card Virtual Channel Extension	Specifies the Remote Desktop Protocol: Smart Card Virtual Channel Extension, an extension (including virtual channels) that supports smart card reader-like devices. Click here to view this version of the [MS-RDPESC] PDF.
[MS-RDPESP]: Remote Desktop Protocol: Serial and Parallel Port Virtual Channel Extension	Specifies the Remote Desktop Protocol: Serial and Parallel Port Virtual Channel Extension, which redirects serial and parallel ports from a terminal client to the terminal server. This extension allows the server to access client ports as if the connected devices were local to the server. Click here to view this version of the [MS-RDPESP] PDF.
[MS-RDPET]: Remote Desktop Protocol: Telemetry Virtual Channel Extension	Specifies the Remote Desktop Protocol: Telemetry Virtual Channel Extension, which extends the Remote Desktop Protocol: Basic Connectivity and Graphics Remoting [MS-RDPBCGR]. This extension is a telemetry protocol that is used to send client performance metrics to the server. Click here to view this version of the [MS-RDPET] PDF.
[MS-RDPETXT]: Remote Desktop Protocol: Text Input Virtual Channel Extension	Specifies the Remote Desktop Protocol: Text Input Virtual Channel Extension, which enables text input and IME integration in virtualized or remote applications. This protocol is used to enable local input methods to operate on applications hosted in virtualized environments or on remote machines. This includes input methods such as speech dictation, software keyboard, IMEs, or handwriting. Click here to view this version of the [MS-RDPETXT] PDF.
[MS-RDPEUDP]: Remote Desktop Protocol: UDP Transport Extension	Specifies the Remote Desktop Protocol: UDP Transport Extension, which extends the transport mechanisms in the Remote Desktop Protocol (RDP) to enable network connectivity between the user's

Specification	Description
	machine and a remote computer system over the User Datagram Protocol (UDP). Click here to view this version of the [MS-RDPEUDP] PDF.
[MS-RDPEUDP2]: Remote Desktop Protocol: UDP Transport Extension Version 2	Remote Desktop Protocol: UDP Transport Extension Version 2 is used to exchange data, for example audio and video, between a remote desktop client and remote desktop server over UDP transport using a URCP based rate control. Click here to view this version of the [MS-RDPEUDP2] PDF.
[MS-RDPEUSB]: Remote Desktop Protocol: USB Devices Virtual Channel Extension	Specifies the Remote Desktop Protocol: USB Devices Virtual Channel Extension, which is used to redirect USB devices from a terminal client to the terminal server. This allows the server access to devices that are physically connected to the client as if the device were local to the server. Click here to view this version of the [MS-RDPEUSB] PDF.
[MS-RDPEV]: Remote Desktop Protocol: Video Redirection Virtual Channel Extension	Specifies the Remote Desktop Protocol: Video Redirection Virtual Channel Extension, which redirects audio/video streams from the terminal server to the terminal client. Click here to view this version of the [MS-RDPEV] PDF.
[MS-RDPEVOR]: Remote Desktop Protocol: Video Optimized Remoting Virtual Channel Extension	Specifies the Remote Desktop Protocol: Video Optimized Remoting Virtual Channel Extension. This is an extension of the Remote Desktop Protocol: Basic Connectivity and Graphics Remoting protocol [MS-RDPBCGR], which runs over a dynamic virtual channel, as specified in [MS-RDPEDYC]. The Remote Desktop Protocol: Video Optimized Remoting Virtual Channel Extension is used to redirect certain rapidly changing graphics content as a video stream from the remote desktop host to the remote desktop client. This protocol specifies the communication between a remote desktop host and a remote desktop client. Click here to view this version of the [MS-RDPEVOR] PDF.
[MS-RDPEWA]: Remote Desktop Protocol: WebAuthn Virtual Channel Protocol	Specifies the Remote Desktop Protocol (RDP): WebAuthn Virtual Channel Protocol which provides a way for a user to do WebAuthn operations over the RDP protocol. It enables a server to send webauthn request to a client, the client can then use this request to talk to authenticators (platform as well as cross-platform) and reply with the response. Click here to view this version of the [MS-RDPEWA] PDF.
[MS-RDPEXPS]: Remote Desktop Protocol: XML	Specifies the Remote Desktop Protocol: XML Paper Specification (XPS) Print Virtual Channel Extension, which redirects printing jobs

Specification	Description
Paper Specification (XPS) Print Virtual Channel Extension	from the terminal server to the terminal client. Click here to view this version of the [MS-RDPEXPS] PDF.
[MS-RDPNSC]: Remote Desktop Protocol: NSCodec Extension	Specifies the Remote Desktop Protocol: NSCodec Extension, an extension to the Remote Desktop Protocol: Basic Connectivity and Graphics Remoting (as specified in [MS-RDPBCGR]). This extension specifies an image codec that can be used to encode screen images by utilizing efficient and effective compression. Click here to view this version of the [MS-RDPNSC] PDF.
[MS-RDPRFX]: Remote Desktop Protocol: RemoteFX Codec Extension	Specifies the Remote Desktop Protocol: RemoteFX Codec Extension, which uses a lossy image codec to encode screen images with efficient and effective compression. Click here to view this version of the [MS-RDPRFX] PDF.
[MS-RDWR]: Remote Desktop Workspace Runtime Protocol	Specifies the Remote Desktop Workspace Runtime Protocol, an HTTP-based protocol for the Remote Desktop Service to discover disconnected sessions for a user and obtain the files required to reconnect to those disconnected sessions. The protocol uses a SOAP-based payload to describe and provide the remote resources to reconnect to a user's disconnected sessions. Click here to view this version of the [MS-RDWR] PDF.
[MS-RMPR]: Rights Management Services (RMS): Client-to-Server Protocol	Specifies the Rights Management Services (RMS) Client-to-Server Protocol, a SOAP protocol used to obtain and issue certificates and licenses used for creating and working with protected content. Click here to view this version of the [MS-RMPR] PDF.
[MS-RMPRS]: Rights Management Services (RMS): Server-to-Server Protocol	Specifies the Rights Management Services (RMS): Server-to-Server Protocol, which is used to communicate information between RMS servers, implementing five interfaces, using either a binary-formatted interface over HTTP or a SOAP-based protocol over HTTP. Click here to view this version of the [MS-RMPRS] PDF.
[MS-RMSI]: Rights Management Services (RMS): ISV Extension Protocol	Specifies the Rights Management Services (RMS): ISV Extension Protocol, a SOAP protocol that is used to communicate information between applications and RMS servers directly without using the RMS client. Click here to view this version of the [MS-RMSI] PDF.
[MS-RNAP]: Vendor-Specific RADIUS Attributes	Specifies the Vendor-Specific RADIUS Attributes for Network Access Protection (NAP) Data Structure protocol, which describes the

Specification	Description
for Network Access Protection (NAP) Data Structure	Microsoft RADIUS vendor-specific attributes (VSAs) that are implemented in the Windows operating system. Click here to view this version of the [MS-RNAP] PDF.
[MS-RNAS]: Vendor-Specific RADIUS Attributes for Network Policy and Access Server Data Structure	Specifies the Vendor-Specific RADIUS Attributes for the Network Policy and Access Server (NPAS) Data Structure protocol, which describes the Microsoft RADIUS vendor-specific attributes (VSAs) that are implemented in the Windows operating system. Click here to view this version of the [MS-RNAS] PDF.
[MS-RPCE]: Remote Procedure Call Protocol Extensions	Specifies the Remote Procedure Call Protocol Extensions, a set of extensions to the DCE Remote Procedure Call 1.1 Specification, as specified in [C706]. These extensions add new capabilities to the DCE 1.1: RPC Specification, allow for more secure implementations to be built, and, in some cases, place additional restrictions on the DCE RPC Specification. Click here to view this version of the [MS-RPCE] PDF.
[MS-RPCH]: Remote Procedure Call over HTTP Protocol	Specifies the Remote Procedure Call over HTTP Protocol, which describes the use of HTTP or HTTPS as a transport for the Remote Procedure Call (RPC) Protocol, as specified in [C706] and extended in [MS-RPCE]. Click here to view this version of the [MS-RPCH] PDF.
[MS-RPCL]: Remote Procedure Call Location Services Extensions	Specifies the Remote Procedure Call Location Services Extensions, a set of extensions and restrictions to the DCE Remote Procedure Call Location Services specification as defined in [C706]. Click here to view this version of the [MS-RPCL] PDF.
[MS-RPRN]: Print System Remote Protocol	Specifies the Print System Remote Protocol, which defines the communication of print job processing and print system management between a print client and a print server. Click here to view this version of the [MS-RPRN] PDF.
[MS-RRASM]: Routing and Remote Access Server (RRAS) Management Protocol	Specifies the Routing and Remote Access Server (RRAS) Management Protocol, which enables remote management (configuration and monitoring) of RRAS. The RRAS implementation refers to the components that can be configured to provide routing, remote access service, and site-to-site connectivity. Click here to view this version of the [MS-RRASM] PDF.
[MS-RRP]: Windows Remote Registry Protocol	Specifies the Windows Remote Registry Protocol, a remote procedure call (RPC)-based client/server protocol that is used to

Specification	Description
	remotely manage a hierarchical data store such as the Windows registry. Click here to view this version of the [MS-RRP] PDF.
[MS-RRSP2]: Remote Rendering Server Protocol Version 2.0	Specifies the Remote Rendering Protocol Version 2, a user interface system for applications in Windows Media Center, which consists of an application-side component model connected to a remote renderer by an asynchronous messaging system that enables the quick and easy construction of captivating interfaces. Click here to view this version of the [MS-RRSP2] PDF.
[MS-RSMC]: Remote Session Monitoring and Control Protocol	Specifies and provides support for client machines to monitor and manage Remote Desktop Protocol (RDP) sessions on a server machine. The protocol provides a set of web service APIs that are implemented as a SOAP-based protocol that uses Hypertext Transfer Protocol (HTTP) and Hypertext Transfer Protocol over Secure Sockets Layer (HTTPS) as its transport. Click here to view this version of the [MS-RSMC] PDF.
[MS-RSMP]: Removable Storage Manager (RSM) Remote Protocol	Specifies the Removable Storage Manager (RSM) Remote Protocol, a set of distributed component object model (DCOM) interfaces for applications to manage robotic changers, media libraries, and tape drives. This protocol deals with detailed low-level operating system and storage concepts. Click here to view this version of the [MS-RSMP] PDF.
[MS-RSP]: Remote Shutdown Protocol	Specifies the Remote Shutdown Protocol, which is designed for shutting down, or for terminating the shutdown, of a remote computer during the shutdown waiting period. Click here to view this version of the [MS-RSP] PDF.
[MS-RSVD]: Remote Shared Virtual Disk Protocol	Specifies the Remote Shared Virtual Disk Protocol, which supports accessing and manipulating virtual disks stored as files on an SMB3 file server. This protocol enables opening, querying, administering, reserving, reading, and writing the virtual disk objects, providing for flexible access by single or multiple consumers. It also provides for forwarding of SCSI operations, to be processed by the virtual disk. Click here to view this version of the [MS-RSVD] PDF.
[MS-RTPDT]: Real-Time Transport Protocol (RTP/RTCP): DTMF Digits, Telephony Tones and	Specifies the Real-Time Transport Protocol (RTP/RTCP): DTMF Digits, Telephony Tones, and Telephony Signals Data Extensions, which describes the payload format needed to carry DTMF digits, tones, and signals in RTP packets over a network transport.

Specification	Description
Telephony Signals Data Extensions	Click here to view this version of the [MS-RTPDT] PDF.
[MS-RTPME]: Real-Time Transport Protocol (RTP/RTCP): Microsoft Extensions	Specifies the Real-Time Transport Protocol (RTP/RTCP): Microsoft Extensions, which is a set of network transport functions suitable for applications transmitting real-time data, such as audio and video, across multimedia endpoints. Click here to view this version of the [MS-RTPME] PDF.
[MS-RTPRAD]: Real-Time Transport Protocol (RTP/RTCP): Redundant Audio Data Extensions	Specifies the Real-Time Transport Protocol (RTP/RTCP): Redundant Audio Data Extensions, which encodes redundant audio data for use with the Real-Time Transport Protocol (RTP) Extensions protocol. Click here to view this version of the [MS-RTPRAD] PDF.
[MS-RTSP]: Real-Time Streaming Protocol (RTSP) Windows Media Extensions	Specifies the Real-Time Streaming Protocol (RTSP) Windows Media Extensions, which defines Windows Media extensions to the Real-Time Streaming Protocol (RTSP). Click here to view this version of the [MS-RTSP] PDF.
[MS-RXAD]: Remote Experience Advertisement Protocol	Specifies the Remote Experience Advertisement Protocol, which enables a Universal Plug and Play (UPnP) service implemented by a device to be used by the client to advertise available remote experience information to that device. Click here to view this version of the [MS-RXAD] PDF.
[MS-SAMLPR]: Security Assertion Markup Language (SAML) Proxy Request Signing Protocol	Specifies the Security Assertion Markup Language (SAML) Proxy Request Signing Protocol, which allows proxy servers to perform operations that require knowledge of configured keys and other state information about federated sites known by the Security Token service server. Click here to view this version of the [MS-SAMLPR] PDF.
[MS-SAMR]: Security Account Manager (SAM) Remote Protocol (Client-to-Server)	Specifies the Security Account Manager (SAM) Remote Protocol, which supports management functionality for an account store or directory containing users and groups. The goal of the protocol is to enable IT administrators and users to manage users, groups, and computers. Click here to view this version of the [MS-SAMR] PDF.
[MS-SAMS]: Security Account Manager (SAM) Remote Protocol (Server-to-Server)	Specifies the Security Account Manager (SAM) Remote Protocol (Server-to-Server). Domain controllers (DCs) use this protocol to forward time-critical database changes to the primary domain controller (PDC), and to forward time-critical database changes

Specification	Description
	from a read-only domain controller (RODC) to a writable NC replica within the same domain outside the normal replication protocol. Click here to view this version of the [MS-SAMS] PDF.
[MS-SCMP]: Shadow Copy Management Protocol	Specifies the Shadow Copy Management Protocol, which programmatically enumerates shadow copies and configures shadow copy storage on remote machines. Click here to view this version of the [MS-SCMP] PDF.
[MS-SCMR]: Service Control Manager Remote Protocol	Specifies the Service Control Manager Remote Protocol, which is used for remotely managing the Service Control Manager (SCM), an RPC server that enables service configuration and control of service programs. Click here to view this version of the [MS-SCMR] PDF.
[MS-SDP]: Session Description Protocol (SDP) Extensions	Specifies the Session Description Protocol (SDP) Extensions, which describes the session description that is used to negotiate instant messaging, audio and video, and data collaboration sessions, and notes the extensions used. Click here to view this version of the [MS-SDP] PDF.
[MS-SFMWA]: Server and File Management Web APIs Protocol	Specifies the Server and File Management Web APIs Protocol, which is used to access a REST-based server and to manage files over the HTTPS transports. The protocol exposes a set of built-in web services for third-party developers to build applications on different devices that can access files and manage servers remotely. The protocol also allows third-party developers to add their own web services without the need to handle authentication. Click here to view this version of the [MS-SFMWA] PDF.
[MS-SFU]: Kerberos Protocol Extensions: Service for User and Constrained Delegation Protocol	Specifies the Kerberos Protocol Extensions: Service for User and Constrained Delegation Protocol, which are two extensions to the Kerberos protocol as developed by Microsoft. These two extensions, collectively known as Service for User (S4U), enable an application service to obtain a Kerberos service ticket on behalf of a user. Click here to view this version of the [MS-SFU] PDF.
[MS-SHLLINK]: Shell Link (.LNK) Binary File Format	Specifies the Shell Link Binary File Format, which contains information that can be used to access another data object. The Shell Link Binary File Format is the format of Windows files with the extension "LNK". Click here to view this version of the [MS-SHLLINK] PDF.

Specification	Description
[MS-SIP]: Session Initiation Protocol Extensions	Specifies Microsoft extensions to the Session Initiation Protocol (SIP), as specified in [RFC3261], which is used by terminals to establish, modify, and terminate multimedia sessions or calls. The SIP extensions add support for privacy features and for subscription requests for offline end nodes to the SIP extensions for presence. Click here to view this version of the [MS-SIP] PDF.
[MS-SMB]: Server Message Block (SMB) Protocol	Specifies the Server Message Block (SMB) Protocol, which defines extensions to the existing Common Internet File System (CIFS) specification that have been implemented by Microsoft since the publication of the [CIFS] specification. Click here to view this version of the [MS-SMB] PDF.
[MS-SMB2]: Server Message Block (SMB) Protocol Versions 2 and 3	Specifies the Server Message Block (SMB) Protocol Versions 2 and 3, which support the sharing of file and print resources between machines and extend the concepts from the Server Message Block Protocol. Click here to view this version of the [MS-SMB2] PDF.
[MS-SMBD]: SMB2 Remote Direct Memory Access (RDMA) Transport Protocol	Specifies the SMB2 Remote Direct Memory Access (RDMA) Transport Protocol, a wrapper for the existing SMB2 protocol that allows SMB2 packets to be delivered over RDMA-capable transports such as iWARP or Infiniband while utilizing the direct data placement (DDP) capabilities of these transports. Benefits include reduced CPU overhead, lower latency, and improved throughput. Click here to view this version of the [MS-SMBD] PDF.
[MS-SMTPNTLM]: NT LAN Manager (NTLM) Authentication: Simple Mail Transfer Protocol (SMTP) Extension	Specifies the NT LAN Manager (NTLM) Authentication: Simple Mail Transfer Protocol (SMTP) Extension, which uses NT LAN Manager (NTLM) authentication (as specified in [MS-NLMP]) by the Simple Mail Transfer Protocol (SMTP) to facilitate client authentication to a Windows SMTP server. Click here to view this version of the [MS-SMTPNTLM] PDF.
[MS-SNID]: Server Network Information Discovery Protocol	Specifies the Server Network Information Discovery Protocol, which defines a pair of request and response messages by which a protocol client can locate protocol servers within the broadcast/multicast scope. The client can then get network information (such as NetBIOS name, Internet Protocol version 4 (IPv4), and Internet Protocol version 6 (IPv6) addresses) about the servers. Click here to view this version of the [MS-SNID] PDF.

Specification	Description
[MS-SNTP]: Network Time Protocol (NTP) Authentication Extensions	Specifies the Network Time Protocol (NTP) Authentication Extensions, which is an authentication extension to the Network Time Protocol (NTP) version 3 ([RFC1305]) and the Simple Network Time Protocol (SNTP) version 4 ([RFC2030]). Click here to view this version of the [MS-SNTP] PDF.
[MS-SPNG]: Simple and Protected GSS-API Negotiation Mechanism (SPNEGO) Extension	Specifies the Simple and Protected GSS-API Negotiation Mechanism (SPNEGO) Protocol Extension. SPNEGO is a security protocol that uses a GSS-API authentication mechanism. GSS-API is a literal set of functions that include both an API and a methodology for approaching authentication. Click here to view this version of the [MS-SPNG] PDF.
[MS-SQMCS]: Software Quality Metrics (SQM) Client-to-Service Version 1 Protocol	Specifies the Software Quality Metrics (SQM) Client-to-Service Protocol V1, used to send software instrumentation metrics to the SQM service and by the client to download client-specific control data. The protocol allows applications and operating system components to collect and send instrumentation metrics to a hosted service. Click here to view this version of the [MS-SQMCS] PDF.
[MS-SQMCS2]: Software Quality Metrics (SQM) Client-to-Service Version 2 Protocol	Specifies the Software Quality Metrics (SQM) Client-to-Service Protocol V2, which is used to send software instrumentation metrics to the SQM service and for the client to download client-specific control data. The protocol extends the concepts of the Software Quality Metrics (SQM) Client-to-Service Protocol, as specified in [MS-SQMCS]. Click here to view this version of the [MS-SQMCS2] PDF.
[MS-SQOS]: Storage Quality of Service Protocol	Specifies the Storage Quality of Service (QoS) Protocol, which is a block-based protocol that is used to manage the Quality of Service configuration of I/O flows targeting remote files accessed over SMB3. Click here to view this version of the [MS-SQOS] PDF.
[MS-SRPL]: Directory Replication Service (DRS) Protocol Extensions for SMTP	Specifies the Directory Replication Service (DRS) Protocol Extensions for SMTP. These are extensions to the DRS Protocol for transport over the Simple Mail Transfer Protocol (SMTP), which provide an alternate transport for the DRS protocol that may allow domain controllers to perform replication in environments where the RPC transport mechanism is unsuitable. Click here to view this version of the [MS-SRPL] PDF.

Specification	Description
[MS-SRVS]: Server Service Remote Protocol	Specifies the Server Service Remote Protocol, which remotely enables file and printer sharing and named pipe access to the server through the Server Message Block Protocol. Click here to view this version of the [MS-SRVS] PDF.
[MS-SSDP]: SSDP: Networked Home Entertainment Devices (NHED) Extensions	Specifies the Networked Home Entertainment Devices (NHED) Extensions, which detects devices on a home network. These extensions provide a mechanism for a control point to discover a device on the network without requiring the device to implement a complete SSDP stack. Click here to view this version of the [MS-SSDP] PDF.
[MS-SSEAN]: Simple Mail Transfer Protocol (SMTP) AUTH Extension for SPNEGO	Specifies the SMTP Service Extension for Negotiate Authentication, which enables SMTP clients to authenticate to SMTP servers by using the Simple and Protected Negotiate (SPNEGO) mechanism. Click here to view this version of the [MS-SSEAN] PDF.
[MS-SSTP]: Secure Socket Tunneling Protocol (SSTP)	Specifies the Secure Socket Tunneling Protocol (SSTP), which is a mechanism to transport data-link layer (L2) frames on a Hypertext Transfer Protocol over Secure Sockets Layer (HTTPS) connection. Click here to view this version of the [MS-SSTP] PDF.
[MS-SSTR]: Smooth Streaming Protocol	Specifies the Smooth Streaming Protocol, which provides a means of delivering media from servers to clients in a way that can be cached by standard HTTP Cache Proxies in the communication chain. Allowing standard HTTP Cache Proxies to respond to requests on behalf of the server increases the number of clients that can be served by a single server. Click here to view this version of the [MS-SSTR] PDF.
[MS-SWN]: Service Witness Protocol	Specifies the Service Witness Protocol, which enables an SMB2 clustered file server to notify SMB2 clients with prompt and explicit notifications about the failure or recovery of a network name and associated services. Click here to view this version of the [MS-SWN] PDF.
[MS-SWSB]: SOAP Over WebSocket Protocol Binding	Specifies the SOAP over WebSocket Protocol Binding, a binding of SOAP to the WebSocket protocol (as defined in [DRAFT-WSP]), including a WSDL transport URI and supported message exchange patterns (MEPs). It specifies how messages defined by a higher-layer protocol are formed and framed for transport over [DRAFT-WSP]. This specification also defines a WebSocket subprotocol. Click here to view this version of the [MS-SWSB] PDF.

Specification	Description
[MS-TAIL]: Telephony API Internet Locator Service Protocol	Specifies the Telephony API Internet Locator Service Protocol, which uses Lightweight Directory Access Protocol (LDAP) requests to retrieve information stored in the Internet Locator Service (ILS) dynamic instance. It is used for communication between a client using the Telephony Application Programming Interface (TAPI) and an ILS server. Click here to view this version of the [MS-TAIL] PDF.
[MS-TCC]: Tethering Control Channel Protocol	Specifies the Tethering Control Channel Protocol, which enables the sharing of the network connection for a server with one or more clients. Click here to view this version of the [MS-TCC] PDF.
[MS-TDS]: Tabular Data Stream Protocol	Specifies the Tabular Data Stream Protocol, which is an application layer request/response protocol that facilitates interaction with a database server and provides for authentication and channel encryption negotiation; specification of requests in SQL (including Bulk Insert); invocation of a stored procedure, also known as a Remote Procedure Call (RPC); returning of data; and Transaction Manager Requests. Click here to view this version of the [MS-TDS] PDF.
[MS-THCH]: Tracing HTTP Correlation Header Protocol	Specifies the Tracing HTTP Correlation Header, which is used to enable correlation between client and server-side traces. Click here to view this version of the [MS-THCH] PDF.
[MS-TIPP]: Transaction Internet Protocol (TIP) Extensions	Specifies the Transaction Internet Protocol (TIP) Extensions, which is a set of extensions to the standard Transaction Internet Protocol (TIP) Version 3.0, as specified in [RFC2371]. The protocol provides concrete mechanisms for associating an OleTx transaction and a TIP transaction. Click here to view this version of the [MS-TIPP] PDF.
[MS-TLSP]: Transport Layer Security (TLS) Profile	Specifies the Transport Layer Security (TLS) Profile, which is the authentication option to the Telnet protocol as a generic method for negotiating an authentication type and mode, including determining whether encryption should be used and whether credentials should be forwarded. Click here to view this version of the [MS-TLSP] PDF.
[MS-TNAP]: Telnet: NT LAN Manager (NTLM) Authentication Protocol	Specifies the Telnet: NT LAN Manager (NTLM) Authentication Protocol, which is the authentication option to the Telnet protocol as a generic method for negotiating an authentication type and

Specification	Description
	mode, including determining whether encryption should be used and whether credentials should be forwarded. Click here to view this version of the [MS-TNAP] PDF.
[MS-TPMVSC]: Trusted Platform Module (TPM) Virtual Smart Card Management Protocol	Specifies the DCOM Interfaces for Trusted Platform Module (TPM) Virtual Smart Card device management, which are used to manage virtual smart cards (VSCs) on a remote machine. They provide methods for a protocol client to request creation and destruction of VSCs, and to monitor the status of these operations. Click here to view this version of the [MS-TPMVSC] PDF.
[MS-TPXS]: Telemetry Protocol XML Schema	Specifies the Telemetry Protocol XML Schema. This schema defines the message structure used by the Software Quality Metrics (SQM) Client-to-Service Protocol V2, specified in [MS-SQMCS2]. The schema is used to send software instrumentation metrics from a client to the SQM service and for the client to download client-specific control data. Click here to view this version of the [MS-TPXS] PDF.
[MS-TRP]: Telephony Remote Protocol	Specifies the Telephony Remote Protocol, which enables implementation of communications applications ranging from voice mail to call centers with multiple agents and switches. Click here to view this version of the [MS-TRP] PDF.
[MS-TSCH]: Task Scheduler Service Remoting Protocol	Specifies the Task Scheduler Service Remoting Protocol, which is used to register and configure a task and to inquire about the status of tasks that are running on a remote machine. Click here to view this version of the [MS-TSCH] PDF.
[MS-TSGU]: Terminal Services Gateway Server Protocol	Specifies the Terminal Services Gateway Server Protocol, which is a mechanism to transport data-link layer (L2) frames on a Hypertext Transfer Protocol over Secure Sockets Layer (HTTPS) connection. Click here to view this version of the [MS-TSGU] PDF.
[MS-TSRAP]: Telnet Server Remote Administration Protocol	Specifies the Telnet Server Remote Administration Protocol, which is a set of interfaces used for performing management tasks on a Telnet Server. Click here to view this version of the [MS-TSRAP] PDF.
[MS-TSTS]: Terminal Services Terminal Server Runtime Interface Protocol	Specifies the Terminal Services Terminal Server Runtime Interface Protocol, which is an RPC-based protocol used for remotely querying and configuring various aspects of a terminal server. Click here to view this version of the [MS-TSTS] PDF.

Specification	Description
[MS-TSWP]: Terminal Services Workspace Provisioning Protocol	Specifies the Terminal Services Workspace Provisioning Protocol, which is used for transferring remote resource information from a server to a client. The client can use this resource information to launch resources such as remote applications on a remote server. Click here to view this version of the [MS-TSWP] PDF.
[MS-TVTT]: Telnet: VTNT Terminal Type Format Data Structure	Specifies the Telnet: VTNT Terminal Type Format Data Structure, which defines the structures for Telnet VTNT Terminal Type Format, and how the client and server negotiate the use of this format. Click here to view this version of the [MS-TVTT] PDF.
[MS-UAMG]: Update Agent Management Protocol	Specifies the Update Agent Management Protocol, which provides a set of types and interfaces that allows callers to manage an update agent and to invoke some update agent operations, such as an update search. Click here to view this version of the [MS-UAMG] PDF.
[MS-UNMP]: User Name Mapping Protocol	Specifies the User Name Mapping Protocol, which maps Windows domain user and group account names to the POSIX user and group identifiers used in AUTH_UNIX authentication, and vice versa. This enables the association of user names for users who have different identities in Windows-based and UNIX-based domains. Click here to view this version of the [MS-UNMP] PDF.
[MS-UPIGD]: UPnP Device and Service Templates: Internet Gateway Device (IGD) Extensions	Specifies the UPnP: Device and Service Templates: Internet Gateway Device (IGD) Extensions. These structure extensions define extensions to the Universal Plug-n-Play (UPnP) device schema that describes an Internet gateway device. Click here to view this version of the [MS-UPIGD] PDF.
[MS-UPMC]: UPnP Device and Service Templates: Media Property and Compatibility Extensions	Specifies the Microsoft Media Property Extensions (MMPE), the Microsoft Compatibility Extension Flags (MCEF), and the Microsoft Power Management Extensions (MPME) to the Universal Plug and Play (UPnP) interoperability guidelines, as specified by the UPnP Forum [UPnP] and used by the Digital Living Network Alliance (DLNA) [DLNA]. Click here to view this version of the [MS-UPMC] PDF.
[MS-USBEPD]: USB Protocol: Platform Detection Extensions	Specifies the USB Protocol: Platform Detection Extensions protocol based on the USB 3.2 specification and Microsoft OS 2.0 Descriptors. It extends the USB protocol with operating system detection to provide OS platform IDs to USB devices so they can take advantage of an operating system's special drivers and features.

Specification	Description
	Click here to view this version of the [MS-USBEPD] PDF.
[MS-V4OF]: IPv4 Over IEEE 1394 Protocol Extensions	Specifies the IPv4 Over IEEE 1394 Protocol Extension, which is the Microsoft extension to the IPv4 over IEEE 1394 protocol to support bridging and clarifies the implementation details as specified in [RFC2734] where necessary. Click here to view this version of the [MS-V4OF] PDF.
[MS-VAPR]: Virtual Application Publishing and Reporting (App-V) Protocol	Specifies the virtual applications that a user is entitled to so that these applications can be downloaded and installed on the user's machine. It is also used to report virtual application usage information to the server so that usage information across multiple users can be aggregated to infer broad virtual application usage patterns across an organization. Click here to view this version of the [MS-VAPR] PDF.
[MS-VDS]: Virtual Disk Service (VDS) Protocol	Specifies the Virtual Disk Service (VDS) Protocol, a set of distributed component object model (DCOM) interfaces for managing the configuration of disk storage. Click here to view this version of the [MS-VDS] PDF.
[MS-VHDX]: Virtual Hard Disk v2 (VHDX) File Format	Specifies the Virtual Hard Disk v2 (VHDX) File Format Protocol, the virtual hard disk format that provides a disk-in-a-file abstraction. Click here to view this version of the [MS-VHDX] PDF.
[MS-VUVP]: VT-UTF8 and VT100+ Protocols	Specifies the VT-UTF8 and VT100+ Protocols, which are used for point-to-point serial communication for terminal control and headless server configuration. Click here to view this version of the [MS-VUVP] PDF.
[MS-W32T]: W32Time Remote Protocol	Specifies the W32Time Remote Protocol, which is used for controlling and monitoring a time service on a machine. This RPC interface supports time services that synchronize time using the Network Time Protocol (NTP) Version 3, as specified in [RFC1305], as well as platform-specific hardware time sources. Click here to view this version of the [MS-W32T] PDF.
[MS-WCCE]: Windows Client Certificate Enrollment Protocol	Specifies the Windows Client Certificate Enrollment Protocol, which consists of a set of DCOM interfaces that enable clients to request various services from a certification authority (CA). These services enable X.509 (as specified in [X509]) digital certificate enrollment, issuance, revocation, and property retrieval. Click here to view this version of the [MS-WCCE] PDF.

Specification	Description
[MS-WCFESAN]: WCF-Based Encrypted Server Administration and Notification Protocol	Specifies the WCF-Based Encrypted Server Administration and Notification Protocol, which enables the protocol client to monitor and manage the protocol server in the same network. Click here to view this version of the [MS-WCFESAN] PDF.
[MS-WDHCE]: Wi-Fi Display Protocol: Hardware Cursor Extension	Specifies the Wi-Fi Display Protocol: Hardware Cursor Extension, which extends the Miracast v1.1 protocol to provide an additional, low-latency stream suitable for controlling the mouse cursor at a higher update rate. Click here to view this version of the [MS-WDHCE] PDF.
[MS-WDSC]: Windows Deployment Services Control Protocol	Specifies the Windows Deployment Services (WDS) Control Protocol, which is an RPC interface that provides the ability to remotely invoke services provided by WDS Server. It is a client/server protocol that uses RPC as a transport. The protocol provides a generic invocation mechanism to send requests to the server and receive replies. Click here to view this version of the [MS-WDSC] PDF.
[MS-WDSMA]: Windows Deployment Services Multicast Application Protocol	Specifies the Windows Deployment Services Multicast Application Protocol, which enables clients to join the multicast session at any point during the lifetime of the .multicast session, and still be able to get all pieces of the content. Click here to view this version of the [MS-WDSMA] PDF.
[MS-WDSMSI]: Windows Deployment Services Multicast Session Initiation Protocol	Specifies the Windows Deployment Services Multicast Session Initiation Protocol, which describes two mechanisms for the client to request initiation of a Multicast Session from the server. Click here to view this version of the [MS-WDSMSI] PDF.
[MS-WDSMT]: Windows Deployment Services Multicast Transport Protocol	Specifies the Windows Deployment Services Multicast Transport Protocol, which enables transmission of content to multiple clients using Multicast UDP. Click here to view this version of the [MS-WDSMT] PDF.
[MS-WDSOSD]: Windows Deployment Services Operation System Deployment Protocol	Specifies the Windows Deployment Services Operation System Deployment Protocol . This protocol defines services exposed by the WDS Server that are used by the clients to deploy an operating system on a machine. Click here to view this version of the [MS-WDSOSD] PDF.
[MS-WDV]: Web Distributed Authoring and	Specifies the Web Distributed Authoring and Versioning (WebDAV) Protocol: Client Extensions, which extends WebDAV by introducing

Specification	Description
Versioning (WebDAV) Protocol: Client Extensions	new headers that both enable the file types that are not currently manageable and optimize protocol interactions for file system clients. These extensions do not introduce new functionality into WebDAV, but instead optimize processing and eliminate the need for special-case processing. Click here to view this version of the [MS-WDV] PDF.
[MS-WDVSE]: Web Distributed Authoring and Versioning (WebDAV) Protocol: Server Extensions	Specifies the Web Distributed Authoring and Versioning (WebDAV) Protocol: Server Extension, which extends the standard HTTP mechanisms defined in [RFC2068] to provide file access and content management over the Internet. Click here to view this version of the [MS-WDVSE] PDF.
[MS-WFDAA]: Wi-Fi Direct (WFD) Application to Application Protocol	Specifies the Wi-Fi Direct (WFD) Protocol: Proximity Extensions, which enable two or more devices that are running the same application to establish a direct connection without requiring an intermediary, such as an infrastructure wireless access point (WAP). Click here to view this version of the [MS-WFDAA] PDF.
[MS-WFDPE]: Wi-Fi Display Protocol Extension	Specifies an extension for the Wi-Fi Display Technical Specification v1.1. Enables latency control, extended diagnostic information, and dynamic format changes on Wi-Fi Display Devices. When implemented, these extensions provide an improved and more consistent Wi-Fi Display experience for a variety of wireless display scenarios, including word processing, web browsing, gaming, and video projection. Click here to view this version of the [MS-WFDPE] PDF.
[MS-WFIM]: Workflow Instance Management Protocol	Specifies the Workflow Instance Management Protocol, which defines a set of SOAP messages for the management of workflow instances, such as suspending, resuming, or canceling an instance. Click here to view this version of the [MS-WFIM] PDF.
[MS-WINSRA]: Windows Internet Naming Service (WINS) Replication and Autodiscovery Protocol	Specifies the Windows Internet Naming Service (WINS) Replication and Autodiscovery Protocol, the Microsoft implementation of NetBIOS Name Server (NBNS). This protocol supports resolution of NetBIOS names to IPv4 addresses. Click here to view this version of the [MS-WINSRA] PDF.
[MS-WKST]: Workstation Service Remote Protocol	Specifies the Workstation Service Remote Protocol, which remotely queries and configures certain aspects of a Server Message Block network redirector on a remote computer. Click here to view this version of the [MS-WKST] PDF.

Specification	Description
[MS-WMF]: Windows Metafile Format	Specifies the Windows Metafile Format structure. A Windows metafile is a container for an image, which is defined by series of variable-length records, called metafile records. Click here to view this version of the [MS-WMF] PDF.
[MS-WMHTTP]: Windows Media HTTP Push Distribution Protocol	Specifies the Windows Media HTTP Push Distribution Protocol, which is used for transferring real-time multimedia data (for example, audio and video) from a client to a server. Click here to view this version of the [MS-WMHTTP] PDF.
[MS-WMI]: Windows Management Instrumentation Remote Protocol	Specifies the Windows Management Instrumentation Remote Protocol, which uses the Common Information Model (CIM), as specified in [DMTF-DSP004], to represent various components of the operating system. CIM is the conceptual model for storing enterprise management information. Click here to view this version of the [MS-WMI] PDF.
[MS-WMIO]: Windows Management Instrumentation Encoding Version 1.0 Protocol	Specifies the Windows Management Instrumentation Encoding Version 1.0 Protocol, which is a binary data encoding format used by the Windows Management Instrumentation Remote Protocol, as specified in [MS-WMI], for network communication. Click here to view this version of the [MS-WMIO] PDF.
[MS-WMLOG]: Windows Media Log Data Structure	Specifies the Windows Media Log Data Structure, which is a syntax for logging messages. The logging messages specify information about how a client received multimedia content from a streaming server. Click here to view this version of the [MS-WMLOG] PDF.
[MS-WMSP]: Windows Media HTTP Streaming Protocol	Specifies the Windows Media HTTP Streaming Protocol, a client/server-based protocol used to stream real-time data between the client (the receiver of streaming data) and server (the sender of streaming data). Click here to view this version of the [MS-WMSP] PDF.
[MS-WPRN]: Web Point-and-Print Protocol	Specifies the Web Point-and-Print Protocol, which is an HTTP-based protocol that clients use to download printer driver software from a server in the client network or from a Web site. This enables distribution of printer driver software using standard Web technologies. Click here to view this version of the [MS-WPRN] PDF.

Specification	Description
[MS-WSDS]: WS-Enumeration: Directory Services Protocol Extensions	Specifies the WS-Enumeration Directory Services Protocol Extensions, a set of extensions to the Web Services Enumeration (WS-Enumeration) [WSENUM] protocol for facilitating SOAP-based search operations against directory servers. Click here to view this version of the [MS-WSDS] PDF.
[MS-WSH]: Windows Security Health Agent (WSHA) and Windows Security Health Validator (WSHV) Protocol	Specifies the Windows Security Health Agent (WSHA) and Windows Security Health Validator (WSHV) Protocol, which reports the system security health state. Click here to view this version of the [MS-WSH] PDF.
[MS-WSMAN]: Web Services Management Protocol Extensions for Windows Server 2003	Specifies the Web Services Management Protocol Extensions, which is a general purpose, SOAP-based systems management extension that defines procedures for carrying out remote management operations. Click here to view this version of the [MS-WSMAN] PDF.
[MS-WSMV]: Web Services Management Protocol Extensions for Windows Vista	Specifies the Web Services Management Protocol Extensions for Windows Vista, which provides Windows Vista extensions to the WS-Management Protocol, the WS-Management Binding Specification, and the WS-CIM Mapping Specification for accessing CIM objects as a Web service. Click here to view this version of the [MS-WSMV] PDF.
[MS-WSP]: Windows Search Protocol	Specifies the Windows Search Protocol (WSP), which allows a client to communicate with a server hosting a Windows Search service (WSS) to issue queries. Click here to view this version of the [MS-WSP] PDF.
[MS-WSPE]: WebSocket Protocol Extensions	Specifies the WebSocket Protocol: Disable Masking Extension, which extends the WebSocket Protocol to improve performance by allowing developers to set a property to disable masking. Click here to view this version of the [MS-WSPE] PDF.
[MS-WSPELD]: WS-Transfer and WS-Enumeration Protocol Extension for Lightweight Directory Access Protocol v3 Controls	Specifies the WS-Transfer: Lightweight Directory Access Protocol (LDAP) v3 Controls, also known as WSPELD. This protocol extends the Web Services Enumeration (WS-Enumeration) [WSENUM] and Web Services Transfer (WS-Transfer) [WXFR] protocols. Click here to view this version of the [MS-WSPELD] PDF.

Specification	Description
[MS-WSPOL]: Web Services: Policy Assertions and WSDL Extensions	Specifies a collection of Web service policy assertions, which define domain-specific behavior for the interaction between two Web service entities. Click here to view this version of the [MS-WSPOL] PDF.
[MS-WSRM]: Windows System Resource Manager (WSRM) Protocol	Specifies the Windows System Resource Manager (WSRM) Protocol, a set of Distributed Component Object Model (DCOM) interfaces for managing the configuration of processor, memory resources, and accounting functions on a server. Click here to view this version of the [MS-WSRM] PDF.
[MS-WSRVCAT]: WS-AtomicTransaction (WS-AT) Version 1.0 Protocol Extensions	Specifies the WS-AtomicTransaction (WS-AT) Version 1.0 Protocol Extensions, which extends the WS-AtomicTransaction protocol by enabling WS-AtomicTransaction initiators, participants, and coordinators to participate in transactions coordinated by OleTx transaction managers. Click here to view this version of the [MS-WSRVCAT] PDF.
[MS-WSRVCRM]: WS-ReliableMessaging Protocol: Advanced Flow Control Extension	Specifies the WS-ReliableMessaging Protocol: Advanced Flow Control Extension, which is an advanced message flow control extension to the Web Services Reliable Messaging Protocol [WSRM1-0] [WSRM1-1]. Click here to view this version of the [MS-WSRVCRM] PDF.
[MS-WSRVCRR]: WS-ReliableMessaging Protocol: Reliable Request-Reply Extension	Specifies the WS-ReliableMessaging Protocol: Reliable Request-Reply Extension. This extension assumes the use of duplex underlying protocols in order to provide support for applications designed to interact using a request-response message exchange pattern. The request-reply extension enables these applications to communicate reliably over transfer protocols that support only SOAP Request-Response. Click here to view this version of the [MS-WSRVCRR] PDF.
[MS-WSTC]: WS-Discovery: Termination Criteria Protocol Extensions	Specifies the WS-Discovery: Termination Criteria Protocol Extensions. This extends the WS-Discovery protocol for sending and receiving termination criteria as part of WS-Discovery Probe and Resolve messages. Click here to view this version of the [MS-WSTC] PDF.
[MS-WSTEP]: WS-Trust X.509v3 Token Enrollment Extensions	Specifies the WS-Trust X.509v3 Token Enrollment Extensions, also known as WSTEP. The protocol specification defines the message formats and server behavior for the purposes of certificate enrollment.

Specification	Description
	Click here to view this version of the [MS-WSTEP] PDF.
[MS-WSTIM]: WS-Transfer: Identity Management Operations for Directory Access Extensions	Specifies the WS-Transfer: Identity Management Operations for Directory Access Extensions, a set of extensions to the WS-Transfer protocol [WXFR] for representing the protocol operations commonly used for directory access in identity management protocols. Click here to view this version of the [MS-WSTIM] PDF.
[MS-WSUSAR]: Windows Server Update Services: Administrative API Remoting Protocol (WSUSAR)	Specifies the Windows Server Update Services: Administrative API Remoting Protocol (WSUSAR), which enables communication between the Windows Server Update Services (WSUS) management API and a WSUS server. Click here to view this version of the [MS-WSUSAR] PDF.
[MS-WSUSSS]: Windows Update Services: Server-Server Protocol	Specifies the Windows Update Services: Server-Server Protocol, which enables a hierarchically organized collection of servers to synchronize metadata and content associated with software updates over the Internet by using SOAP and HTTP protocols. Click here to view this version of the [MS-WSUSSS] PDF.
[MS-WUSP]: Windows Update Services: Client-Server Protocol	Specifies the Windows Update Services: Client-Server Protocol, which enables machines to discover and download software updates over the Internet using the SOAP and HTTP protocols. Click here to view this version of the [MS-WUSP] PDF.
[MS-XCA]: Xpress Compression Algorithm	Specifies the three variants of the Xpress Compression Algorithm: LZ77+Huffman, Plain LZ77, LZNT1, and their respective decompression algorithms. This algorithm efficiently compresses data that contains repeated byte sequences. It is not designed to compress image, audio, or video data. Between the trade-offs of compressed size and CPU cost, it heavily emphasizes low CPU cost. Click here to view this version of the [MS-XCA] PDF.
[MS-XCEP]: X.509 Certificate Enrollment Policy Protocol	Specifies the X.509 Certificate Enrollment Policy Protocol. This protocol defines the interactions between a requesting client and a responding server for the exchange of a certificate enrollment policy, which is the collection of certificate templates and certificate issuers available to the requestor for X.509 certificate enrollment. Click here to view this version of the [MS-XCEP] PDF.
[MS-XOPP]: XML-binary Optimized Packaging (XOP) Profile	Specifies the XML-binary Optimized Packaging (XOP) Profile, which provides extensions that enable more efficient implementations of

Specification	Description
	[XML-XOP] to be built by requiring certain ordering of the MIME parts in the XOP package Click here to view this version of the [MS-XOPP] PDF.
[MS-XUSBI]: Xbox Universal Serial Bus Protocol (XUSB) Interface Extension	Specifies the Xbox Universal Serial Bus Protocol (XUSB) Interface Extension of the USB 2.0 interface that provides extended semantics for interaction between game controller devices and a host. This protocol includes enumeration of device capabilities, determining of device type and subtype, transfer of gamepad and voice data, and support for an expansion device on the controller. Click here to view this version of the [MS-XUSBI] PDF.

[MS-FSCC]: File System Control Codes

Article04/07/2025

Specifies the File System Control Codes that define the network format of native Windows structures that may be used within other protocols.

This page and associated content may be updated frequently. We recommend you subscribe to the RSS feed to receive update notifications.

Published Version

Expand table

Date	Protocol Revision	Revision Class	Downloads
4/7/2025	58.0	Major	PDF | DOCX

Click here to download a zip file of all PDF files for Windows Protocols.

Previous Versions

Expand table

Date	Protocol Revision	Revision Class	Downloads
2/10/2025	57.0	Major	PDF | DOCX
10/7/2024	56.0	Major	PDF | DOCX
9/16/2024	55.0	None	PDF | DOCX
7/8/2024	55.0	Major	PDF | DOCX | Diff
4/23/2024	54.0	Major	PDF | DOCX | Diff
9/20/2023	53.0	Major	PDF | DOCX | Diff
4/29/2022	52.0	Major	PDF | DOCX | Errata | Diff
6/25/2021	51.0	Major	PDF | DOCX | Errata | Diff
6/2/2021	50.0	Major	PDF | DOCX | Diff
4/7/2021	49.0	Major	PDF | DOCX | Diff
8/26/2020	48.0	Major	PDF | DOCX | Errata | Diff

Date	Protocol Revision	Revision Class	Downloads
3/4/2020	47.0	Major	PDF	DOCX	Errata	Diff
9/23/2019	46.0	Major	PDF	DOCX	Errata	Diff
9/12/2018	45.0	Major	PDF	DOCX	Errata	Diff
3/16/2018	44.0	Major	PDF	DOCX	Errata	Diff
12/1/2017	43.0	Major	PDF	DOCX	Diff
9/15/2017	42.0	Major	PDF	DOCX	Errata	Diff
6/1/2017	41.0	Major	PDF	DOCX	Errata	Diff
7/14/2016	40.0	Major	PDF	DOCX	Errata	Diff
10/16/2015	39.0	Major	PDF	DOCX	Errata
6/30/2015	38.0	Major	PDF	DOCX	Errata
5/15/2014	37.0	None	PDF	DOCX
2/13/2014	37.0	Major	PDF	DOCX
11/14/2013	36.0	None	PDF	DOCX
8/8/2013	36.0	Major	PDF	DOCX
1/31/2013	35.0	Major	
10/25/2012	34.0	Major	
7/12/2012	33.0	Major	
3/30/2012	32.0	Major	
12/16/2011	31.0	Major	
9/23/2011	30.2	Minor	
6/17/2011	30.1	Minor	
5/6/2011	30.0	Major	
3/25/2011	29.0	Major	
2/11/2011	28.0	Major	
1/7/2011	27.1	None	
11/19/2010	27.1	Minor	

Date	Protocol Revision	Revision Class	Downloads
10/8/2010	27.0	Major	
8/27/2010	26.0	Major	
7/16/2010	25.0	Major	
6/4/2010	24.0	Major	
4/23/2010	23.0	Major	
3/12/2010	22.0	Major	
1/29/2010	21.0	Major	
12/18/2009	20.0	Major	
11/6/2009	19.0	Major	
9/25/2009	18.0	Major	
8/14/2009	17.0	Major	
7/2/2009	16.0	Major	
5/22/2009	15.0	Major	
4/10/2009	14.0	Major	
2/27/2009	13.0	Major	
1/16/2009	12.0	Major	
12/5/2008	11.0	Major	
10/24/2008	10.0	Major	
8/29/2008	9.0	Major	
7/25/2008	8.0	Major	
6/20/2008	7.0	Major	
5/16/2008	6.0	Major	
3/14/2008	5.0.3	Editorial	
1/25/2008	5.0.2	Editorial	
11/30/2007	5.0.1	Editorial	
10/23/2007	5.0	Major	

Date	Protocol Revision	Revision Class	Downloads
9/28/2007	4.0	Major	
8/10/2007	3.0	Major	
7/20/2007	2.0	Major	
7/3/2007	1.0	Major	
4/3/2007	0.01	New	

Preview Versions

From time to time, Microsoft may publish a preview, or pre-release, version of an Open Specifications technical document for community review and feedback. To submit feedback for a preview version of a technical document, please follow any instructions specified for that document. If no instructions are indicated for the document, please provide feedback by using the Open Specification Forums.

The preview period for a technical document varies. Additionally, not every technical document will be published for preview.

A preview version of this document may be available on the Windows Protocols - Preview Documents page. After the preview period, the most current version of the document is available on this page.

Development Resources

Find resources for creating interoperable solutions for Microsoft software, services, hardware, and non-Microsoft products:

Plugfests and Events, Test Tools, Development Support, and Open Specifications Dev Center.

Intellectual Property Rights Notice for Open Specifications Documentation

- **Technical Documentation.** Microsoft publishes Open Specifications documentation ("this documentation") for protocols, file formats, data portability, computer languages, and standards support. Additionally, overview documents cover inter-protocol relationships and interactions.

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

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

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

- **License Programs**. To see all of the protocols in scope under a specific license program and the associated patents, visit the Patent Map.

- **Trademarks**. The names of companies and products contained in this documentation might be covered by trademarks or similar intellectual property rights. This notice does not grant any licenses under those rights. For a list of Microsoft trademarks, visit www.microsoft.com/trademarks.

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

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

Tools. The Open Specifications documentation does not require the use of Microsoft programming tools or programming environments in order for you to develop an implementation. If you have access to Microsoft programming tools and environments, you are free to take advantage of them. Certain Open Specifications documents are intended for use in conjunction with publicly available standards specifications and network programming art and,

as such, assume that the reader either is familiar with the aforementioned material or has immediate access to it.

Support. For questions and support, please contact dochelp@microsoft.com.

1 Introduction

This specification defines the network format of native Windows structures that can 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.

Sections 1.7 and 2 of this specification are normative. All other sections and examples in this specification are informative.

Last updated on 11/21/2025

1.1 Glossary

Article05/10/2022

This document uses the following terms:

8.3 name: A file name string restricted in length to 12 characters that includes a base name of up to eight characters, one character for a period, and up to three characters for a file name extension. For more information on 8.3 file names, see [MS-CIFS] section 2.2.1.1.1.

access control list (ACL): A list of access control entries (ACEs) that collectively describe the security rules for authorizing access to some resource; for example, an object or set of objects.

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.

binary large object (BLOB): A collection of binary data stored as a single entity in a database.

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

cluster: The smallest allocation unit on a volume.

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

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

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

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

Distributed Link Tracking (DLT): A protocol that enables client applications to track sources that have been sent to remote locations using remote procedure call (RPC) interfaces, and to maintain links to files. It exposes methods that belong to two

interfaces, one of which exists on the server (trksvr) and the other on a workstation (trkwks).

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

FAT file system: A file system used to organize and manage files. The file allocation table (FAT) is a data structure that the operating system creates when a volume is formatted by using FAT or FAT32 file systems. The operating system stores information about each file in the FAT so that it can retrieve the file later.

Fid: A 16-bit value that the Server Message Block (SMB) server uses to represent an opened file, named pipe, printer, or device. A Fid is returned by an SMB server in response to a client request to open or create a file, named pipe, printer, or device. The SMB server guarantees that the Fid value returned is unique for a given SMB connection until the SMB connection is closed, at which time the Fid value can be reused. The Fid is used by the SMB client in subsequent SMB commands to identify the opened file, named pipe, printer, or device.

file allocation table (FAT): A data structure that the operating system creates when a volume is formatted by using FAT or FAT32 file systems. The operating system stores information about each file in the FAT so that it can retrieve the file later.

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

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

file stream: See main stream and named stream.

file system control (FSCTL): A command issued to a file system to alter or query the behavior of the file system and/or set or query metadata that is associated with a particular file or with the file system itself.

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

filter manager: A file system filter driver that simplifies the development of other file system filter drivers. Although it is possible to write a filter driver that manages other filters, for the purposes of this document, the phrase filter manager refers only to the file

system filter manager, which is an operating system component. A filter driver developed to the filter manager model is called a minifilter.

globally unique identifier (GUID): A term used interchangeably with universally unique identifier (UUID) in Microsoft protocol technical documents (TDs). Interchanging the usage of these terms does not imply or require a specific algorithm or mechanism to generate the value. Specifically, the use of this term does not imply or require that the algorithms described in [RFC4122] or [C706] must be used for generating the GUID. See also universally unique identifier (UUID).

GUIDString: A GUID in the form of an ASCII or Unicode string, consisting of one group of 8 hexadecimal digits, followed by three groups of 4 hexadecimal digits each, followed by one group of 12 hexadecimal digits. It is the standard representation of a GUID, as described in [RFC4122] section 3. For example, "6B29FC40-CA47-1067-B31D-00DD010662DA". Unlike a curly braced GUID string, a GUIDString is not enclosed in braces.

I/O control (IOCTL): A command that is issued to a target file system or target device in order to query or alter the behavior of the target; or to query or alter the data and attributes that are associated with the target or the objects that are exposed by the target.

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

logical cluster number (LCN): The cluster number relative to the beginning of the volume. The first cluster on a volume is zero (0).

mailslot: A mechanism for one-way interprocess communications (IPC). For more information, see [MSLOT] and [MS-MAIL].

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

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

named stream: A place within a file in addition to the main stream where data is stored, or the data stored therein. File systems support a mode in which it is possible to open either the main stream of a file and/or to open a named stream. Named streams have different data than the main stream (and than each other) and can be read and written independently. Not all file systems support named streams. See also main stream.

NetBIOS name: A 16-byte address that is used to identify a NetBIOS resource on the network. For more information, see [RFC1001] and [RFC1002].

NT file system (NTFS): A proprietary Microsoft file system. For more information, see [MSFT-NTFS].

Object ID: See ObjectID.

object identifier (OID): In the context of an object server, a 64-bit number that uniquely identifies an object.

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

Offload Read: A variant to a normal read operation where a target device generates and returns a Token instead of a buffer containing the data to be read. The Token is maintained by the target device until it invalidates the Token for any vendor-specific reason. The data logically represented by the Token cannot change, and the target device is required to maintain this representation. An example of a target device is a SAN Storage Array with support for the associated low-level storage commands. For more information on Offload Read, see [INCITS-T10/11-059].

Offload Write: A variant to a normal write operation where the host provides a Token instead of a buffer containing the data to be written. Upon receipt of the Offload Write, the target device parses the Token and determines whether the data movement (the Write) can be completed to the requested location. An example of a target device is a SAN Storage Array with support for the associated low-level storage commands. For more information on Offload Write, see [INCITS-T10/11-059].

reparse point: An attribute that can be added to a file to store a collection of user-defined data that is opaque to NTFS or ReFS. If a file that has a reparse point is opened, the open will normally fail with STATUS_REPARSE, so that the relevant file system filter driver can detect the open of a file associated with (owned by) this reparse point. At that point, each installed filter driver can check to see if it is the owner of the reparse point, and, if so, perform any special processing required for a file with that reparse point. The format of this data is understood by the application that stores the data and the file system filter that interprets the data and processes the file. For example, an encryption filter that is marked as the owner of a file's reparse point could look up the encryption key for that file. A file can have (at most) 1 reparse point associated with it. For more information, see [MS-FSCC].

reparse point tag: A unique identifier for a file system filter driver stored within a file's optional reparse point data that indicates the file system filter driver that performs

additional filter-defined processing on a file during I/O operations. An implementer can 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] .

replica set: In File Replication Service (FRS), the replication of files and directories according to a predefined topology and schedule on a specific folder. The topology and schedule are collectively called a replica set. A replica set contains a set of replicas, one for each machine that participates in replication.

sector: The smallest addressable unit of a disk.

security identifier (SID): An identifier for security principals that is used to identify an account or a group. Conceptually, the SID is composed of an account authority portion (typically a domain) and a smaller integer representing an identity relative to the account authority, termed the relative identifier (RID). The SID format is specified in [MS-DTYP] section 2.4.2; a string representation of SIDs is specified in [MS-DTYP] section 2.4.2 and [MS-AZOD] section 1.1.1.2.

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

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.

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

stream: A sequence of bytes written to a file on the target file system. Every file stored on a volume that uses the file system contains at least one stream, which is normally used to store the primary contents of the file. Additional streams within the file can be used to store file attributes, application parameters, or other information specific to that file. Every file has a default data stream, which is unnamed by default. That data stream, and any other data stream associated with a file, can optionally be named.

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

symbolic link: A symbolic link is a reparse point that points to another file system object. The object being pointed to is called the target. Symbolic links are transparent to users; the links appear as normal files or directories, and can be acted upon by the user or application in exactly the same manner. Symbolic links can be created using the FSCTL_SET_REPARSE_POINT request as specified in [MS-FSCC] section 2.3.61. They can be deleted using the FSCTL_DELETE_REPARSE_POINT request as specified in [MS-FSCC] section 2.3.5. Implementing symbolic links is optional for a file system.

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

token: A 512-byte length opaque string that is generated and maintained by a supported target device. A Token functions logically as an immutable point-in-time representation for a set of data specified by a host and can be conceptualized as a compressed representation of the data that only a certain class of storage subsystems can interpret. A Token can also be constructed from a set of well-known Tokens to enable the client to describe a homogeneous attribute for a set of data (for example, all zeros) or to enable a server to apply a homogeneous attribute to a set of data (for example, a set of all zeros). For more information on Tokens, see [INCITS-T10/11-059].

Unicode character: Unless otherwise specified, a 16-bit UTF-16 code unit.

Uniform Resource Locator (URL): A string of characters in a standardized format that identifies a document or resource on the World Wide Web. The format is as specified in [RFC1738].

Universal Disk Format (UDF): A type of file system for storing files on optical media.

update sequence number (USN): The offset from the beginning of the change journal stream that uniquely identifies a change journal record.

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.

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

1.2 References

Article02/14/2019

Links to a document in the Microsoft Open Specifications library point to the correct section in the most recently published version of the referenced document. However, because individual documents in the library are not updated at the same time, the section numbers in the documents may not match. You can confirm the correct section numbering by checking the Errata.

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.

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

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

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

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

[MS-RDPBCGR] Microsoft Corporation, "Remote Desktop Protocol: Basic Connectivity and Graphics Remoting".

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

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

[MS-SQLRS] Microsoft Corporation, "SQL Server Remote Storage Profile".

[RFC1094] Sun Microsystems, Inc., "NFS: Network File System Protocol Specification", RFC 1094, March 1989, https://www.rfc-editor.org/info/rfc1094

[RFC1813] Callaghan, B., Pawlowski, B., and Staubach, P., "NFS Version 3 Protocol Specification", RFC 1813, June 1995, https://www.rfc-editor.org/info/rfc1813

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

Last updated on 11/21/2025

1.2.2 Informative References

Article09/26/2024

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

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

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

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

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

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

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

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

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

[MSDN-SECZONES] Microsoft Corporation, "About URL Security Zones", http://msdn.microsoft.com/en-us/library/ms537183.aspx

[MSFT-NTFSWorks] Microsoft Corporation, "How NTFS Works", March 2003, http://technet.microsoft.com/en-us/library/cc781134(WS.10).aspx

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

[MSKB-5014019] Microsoft Corporation, "KB5014019 May 2022", KB5014019 May 2022, https://support.microsoft.com/en-us/topic/may-24-2022-kb5014019-os-build-22000-708-preview-442dbde4-ce28-4345-aecf-2d4744376418

[MSKB-5014021] Microsoft Corporation, "KB5014021 May 2022", KB5014021 May 2022, https://support.microsoft.com/en-us/topic/may-24-2022-kb5014021-os-build-20348-740-preview-2b180bd4-dceb-4c49-b8cf-402b342ebc84

[MSKB-5014022] Microsoft Corporation, "KB5014022 May 2022", KB5014022 May 2022, https://support.microsoft.com/en-us/topic/may-24-2022-kb5014022-os-build-17763-2989-preview-08f88943-2fc8-4fdb-a13b-ba89af313d06

[MSKB-5014023] Microsoft Corporation, "KB5014023 June 2022", https://support.microsoft.com/en-us/topic/june-2-2022-kb5014023-os-builds-19042-1741-19043-1741-and-19044-1741-preview-65ac6a5d-439a-4e88-b431-a5e2d4e2516a

[MSKB-5014702] Microsoft Corporation, "KB5014702 - June 2022", KB5014702, June 14, 2022, https://support.microsoft.com/en-us/topic/june-14-2022-kb5014702-os-build-14393-5192-e60ac0e1-44a4-49f9-871f-7c25eb0e5bb1

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

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

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

[UASDC] Ziv, J. and Lempel, A., "A Universal Algorithm for Sequential Data Compression", May 1977, http://www.cs.duke.edu/courses/spring03/cps296.5/papers/ziv_lempel_1977_universal_algorithm.pdf

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

[WHDC-RPTR] Microsoft Corporation, "Reparse Point Tag Request", https://learn.microsoft.com/en-us/windows-hardware/drivers/ifs/reparse-point-tag-request

[WININTERNALS] Russinovich, M., and Solomon, D., "Microsoft Windows Internals, Fourth Edition", Microsoft Press, 2005, ISBN: 0735619174.

1.3 Overview

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

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

The following topics are addressed in this specification:

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

Last updated on 11/21/2025

1.4 Relationship to Protocols and Other Structures

Article04/23/2024

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

1.5 Applicability Statement

Article09/20/2023

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

1.6 Versioning and Localization

Article04/23/2024

None.

1.7 Vendor-Extensible Fields

Article04/07/2025

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 MUST request a unique reparse tag for that application from Microsoft by following the instructions described in [WHDC-RPTR]. For more information about reparse points, see [REPARSE].

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

2 Structures

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

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

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

Unless otherwise qualified, instances of **GUID** in this section refer to [MS-DTYP] section 2.3.4.

Last updated on 11/21/2025

2.1 Common Data Types

Article04/23/2024

2.1.1 Time

Article08/24/2020

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

See FILETIME ([MS-DTYP] section 2.3.3) for related information.

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

2.1.2 Reparse Point Data Structures

Article02/14/2019

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

2.1.2.1 Reparse Tags

06/10/2025

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

Reparse tags are stored as 32-bit unsigned integer values, as shown in the following diagram.

Expand table

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
M	R	N	D	Reserved												Value															

M (1 bit): Microsoft bit. If this bit is set to 1, the tag is owned by Microsoft. All other tags MUST use zero for this bit.

R (1 bit): Reserved bit. This bit MUST be set to zero for non-Microsoft tags. It was formerly known as High-latency bit.

N (1 bit): Name Surrogate bit. If this bit is set to 1, the file or directory represents another named entity in the system.

D (1 bit): Directory bit. Indicates that any directory with this reparse tag can have children. This bit does not have special meaning when used on a non-directory file. This bit MUST NOT be set when N (Name Surrogate) bit is set.

Reserved (12 bits): This field is reserved. This field SHOULD be set to 0 and MUST be ignored on receipt.

Value (2 bytes): A 16-bit unsigned integer containing the reparse point tag that uniquely identifies the owner of the reparse point.

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

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

Expand table

Value	Meaning
IO_REPARSE_TAG_RESERVED_ZERO 0x00000000	Reserved reparse tag value.
IO_REPARSE_TAG_RESERVED_ONE 0x00000001	Reserved reparse tag value.

Value	Meaning
IO_REPARSE_TAG_RESERVED_TWO 0x00000002	Reserved reparse tag value.
IO_REPARSE_TAG_MOUNT_POINT 0xA0000003	Used for mount point support, specified in section 2.1.2.5.
IO_REPARSE_TAG_HSM 0xC0000004	Obsolete. Used by legacy Hierarchical Storage Management Product.
IO_REPARSE_TAG_DRIVE_EXTENDER 0x80000005	Home server drive extender.<3>
IO_REPARSE_TAG_HSM2 0x80000006	Obsolete. Used by legacy Hierarchical Storage Management Product.
IO_REPARSE_TAG_SIS 0x80000007	Used by single-instance storage (SIS) filter driver. Server-side interpretation only, not meaningful over the wire.
IO_REPARSE_TAG_WIM 0x80000008	Used by the WIM Mount filter. Server-side interpretation only, not meaningful over the wire.
IO_REPARSE_TAG_CSV 0x80000009	Obsolete. Used by Clustered Shared Volumes (CSV) version 1 in Windows Server 2008 R2 operating system. Server-side interpretation only, not meaningful over the wire.
IO_REPARSE_TAG_DFS 0x8000000A	Used by the DFS filter. The DFS is described in the Distributed File System (DFS): Referral Protocol Specification [MS-DFSC]. Server-side interpretation only, not meaningful over the wire.
IO_REPARSE_TAG_FILTER_MANAGER 0x8000000B	Used by filter manager test harness.<4>
IO_REPARSE_TAG_SYMLINK 0xA000000C	Used for symbolic link support. See section 2.1.2.4.
IO_REPARSE_TAG_IIS_CACHE 0xA0000010	Used by Microsoft Internet Information Services (IIS) caching. Server-side interpretation only, not meaningful over the wire.
IO_REPARSE_TAG_DFSR 0x80000012	Used by the DFS filter. The DFS is described in [MS-DFSC]. Server-side interpretation only, not meaningful over the wire.
IO_REPARSE_TAG_DEDUP 0x80000013	Used by the Data Deduplication (Dedup) filter. Server-side interpretation only, not meaningful over the wire.
IO_REPARSE_TAG_APPXSTRM 0xC0000014	Not used.

Value	Meaning
IO_REPARSE_TAG_NFS 0x80000014	Used by the Network File System (NFS) component. Server-side interpretation only, not meaningful over the wire.
IO_REPARSE_TAG_FILE_PLACEHOLDER 0x80000015	Obsolete. Used by Windows Shell for legacy placeholder files in Windows 8.1. Server-side interpretation only, not meaningful over the wire.
IO_REPARSE_TAG_DFM 0x80000016	Used by the Dynamic File filter. Server-side interpretation only, not meaningful over the wire.
IO_REPARSE_TAG_WOF 0x80000017	Used by the Windows Overlay filter, for either WIMBoot or single-file compression. Server-side interpretation only, not meaningful over the wire.
IO_REPARSE_TAG_WCI 0x80000018	Used by the Windows Container Isolation filter. Server-side interpretation only, not meaningful over the wire.
IO_REPARSE_TAG_WCI_1 0x90001018	Used by the Windows Container Isolation filter. Server-side interpretation only, not meaningful over the wire.
IO_REPARSE_TAG_GLOBAL_REPARSE 0xA0000019	Used by NPFS to indicate a named pipe symbolic link from a server silo into the host silo. Server-side interpretation only, not meaningful over the wire.
IO_REPARSE_TAG_CLOUD 0x9000001A	Used by the Cloud Files filter, for files managed by a sync engine such as Microsoft OneDrive. Server-side interpretation only, not meaningful over the wire.
IO_REPARSE_TAG_CLOUD_1 0x9000101A	Used by the Cloud Files filter, for files managed by a sync engine such as OneDrive. Server-side interpretation only, not meaningful over the wire.
IO_REPARSE_TAG_CLOUD_2 0x9000201A	Used by the Cloud Files filter, for files managed by a sync engine such as OneDrive. Server-side interpretation only, not meaningful over the wire.
IO_REPARSE_TAG_CLOUD_3 0x9000301A	Used by the Cloud Files filter, for files managed by a sync engine such as OneDrive. Server-side interpretation only, not meaningful over the wire.
IO_REPARSE_TAG_CLOUD_4 0x9000401A	Used by the Cloud Files filter, for files managed by a sync engine such as OneDrive. Server-side interpretation only, not meaningful over the wire.
IO_REPARSE_TAG_CLOUD_5 0x9000501A	Used by the Cloud Files filter, for files managed by a sync engine such as OneDrive. Server-side interpretation only, not meaningful over the wire.
IO_REPARSE_TAG_CLOUD_6 0x9000601A	Used by the Cloud Files filter, for files managed by a sync engine such as OneDrive. Server-side interpretation only, not meaningful over the wire.
IO_REPARSE_TAG_CLOUD_7 0x9000701A	Used by the Cloud Files filter, for files managed by a sync engine such as OneDrive. Server-side interpretation only, not meaningful over the wire.

Value	Meaning
IO_REPARSE_TAG_CLOUD_8 0x9000801A	Used by the Cloud Files filter, for files managed by a sync engine such as OneDrive. Server-side interpretation only, not meaningful over the wire.
IO_REPARSE_TAG_CLOUD_9 0x9000901A	Used by the Cloud Files filter, for files managed by a sync engine such as OneDrive. Server-side interpretation only, not meaningful over the wire.
IO_REPARSE_TAG_CLOUD_A 0x9000A01A	Used by the Cloud Files filter, for files managed by a sync engine such as OneDrive. Server-side interpretation only, not meaningful over the wire.
IO_REPARSE_TAG_CLOUD_B 0x9000B01A	Used by the Cloud Files filter, for files managed by a sync engine such as OneDrive. Server-side interpretation only, not meaningful over the wire.
IO_REPARSE_TAG_CLOUD_C 0x9000C01A	Used by the Cloud Files filter, for files managed by a sync engine such as OneDrive. Server-side interpretation only, not meaningful over the wire.
IO_REPARSE_TAG_CLOUD_D 0x9000D01A	Used by the Cloud Files filter, for files managed by a sync engine such as OneDrive. Server-side interpretation only, not meaningful over the wire.
IO_REPARSE_TAG_CLOUD_E 0x9000E01A	Used by the Cloud Files filter, for files managed by a sync engine such as OneDrive. Server-side interpretation only, not meaningful over the wire.
IO_REPARSE_TAG_CLOUD_F 0x9000F01A	Used by the Cloud Files filter, for files managed by a sync engine such as OneDrive. Server-side interpretation only, not meaningful over the wire.
IO_REPARSE_TAG_APPEXECLINK 0x8000001B	Used by Universal Windows Platform (UWP) packages to encode information that allows the application to be launched by CreateProcess. Server-side interpretation only, not meaningful over the wire.
IO_REPARSE_TAG_PROJFS 0x9000001C	Used by the Windows Projected File System filter, for files managed by a user mode provider such as VFS for Git. Server-side interpretation only, not meaningful over the wire.
IO_REPARSE_TAG_LX_SYMLINK 0xA000001D	Used by the Windows Subsystem for Linux (WSL) to represent a UNIX symbolic link. section 2.1.2.7.
IO_REPARSE_TAG_STORAGE_SYNC 0x8000001E	Used by the Azure File Sync (AFS) filter. Server-side interpretation only, not meaningful over the wire.
IO_REPARSE_TAG_STORAGE_SYNC_FOLDER 0x90000027	Used by the Azure File Sync (AFS) filter for folder. Server-side interpretation only, not meaningful over the wire.
IO_REPARSE_TAG_WCI_TOMBSTONE 0xA000001F	Used by the Windows Container Isolation filter. Server-side interpretation only, not meaningful over the wire.
IO_REPARSE_TAG_UNHANDLED 0x80000020	Used by the Windows Container Isolation filter. Server-side interpretation only, not meaningful over the wire.

Value	Meaning
IO_REPARSE_TAG_ONEDRIVE 0x80000021	Not used.
IO_REPARSE_TAG_PROJFS_TOMBSTONE 0xA0000022	Used by the Windows Projected File System filter, for files managed by a user mode provider such as VFS for Git. Server-side interpretation only, not meaningful over the wire.
IO_REPARSE_TAG_AF_UNIX 0x80000023	Used to represent a UNIX domain socket. Server-side interpretation only, not meaningful over the wire. No defined structure.
IO_REPARSE_TAG_LX_FIFO 0x80000024	Used by the Windows Subsystem for Linux (WSL) to represent a UNIX FIFO (named pipe). Server-side interpretation only, not meaningful over the wire. No defined structure.
IO_REPARSE_TAG_LX_CHR 0x80000025	Used by the Windows Subsystem for Linux (WSL) to represent a UNIX character special file. Server-side interpretation only, not meaningful over the wire. No defined structure.
IO_REPARSE_TAG_LX_BLK 0x80000026	Used by the Windows Subsystem for Linux (WSL) to represent a UNIX block special file. Server-side interpretation only, not meaningful over the wire. No defined structure.
IO_REPARSE_TAG_WCI_LINK 0xA0000027	Used by the Windows Container Isolation filter. Server-side interpretation only, not meaningful over the wire.
IO_REPARSE_TAG_WCI_LINK_1 0xA0001027	Used by the Windows Container Isolation filter. Server-side interpretation only, not meaningful over the wire.

2.1.2.2 REPARSE_DATA_BUFFER

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

This data element has the following subtypes:

- Symbolic Link Reparse Data Buffer
- Mount Point Reparse Data Buffer
- Network File System (NFS) Reparse Data Buffer
- LX SYMLINK REPARSE_DATA_BUFFER

Expand table

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
ReparseTag																															
ReparseDataLength																Reserved															
DataBuffer (variable)																															
...																															

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

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

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

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

Last updated on 11/21/2025

2.1.2.3 REPARSE_GUID_DATA_BUFFER

Article04/07/2025

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

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

Expand table

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
ReparseTag																															
ReparseDataLength																Reserved															
ReparseGuid (16 bytes)																															
...																															
...																															
DataBuffer (variable)																															
...																															

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

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

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

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

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

2.1.2.4 Symbolic Link Reparse Data Buffer

Article04/23/2024

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

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

Expand table

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
ReparseTag																															
ReparseDataLength																														Reserved	
SubstituteNameOffset																														SubstituteNameLength	
PrintNameOffset																														PrintNameLength	
Flags																															
PathBuffer (variable)																															
...																															

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

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

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

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

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

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

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

Expand table

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

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

2.1.2.5 Mount Point Reparse Data Buffer

Article12/14/2021

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

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

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
ReparseTag																															
ReparseDataLength																Reserved															
SubstituteNameOffset																SubstituteNameLength															
PrintNameOffset																PrintNameLength															
PathBuffer (variable)																															
...																															

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

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

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

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

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

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

2.1.2.6 Network File System (NFS) Reparse Data Buffer

Article10/30/2020

The Network File System Reparse Data Buffer data element is a subtype of REPARSE_DATA_BUFFER, which contains information about symbolic files and devices created by the Network File System client.

0	1	2	3	4	5	6	7	8	9	10	1	2	3	4	5	6	7	8	9	20	1	2	3	4	5	6	7	8	9	30	1
ReparseTag																															
ReparseDataLength																Reserved															
GenericReparseBuffer (variable)																															
...																															

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 0x80000014.

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

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

GenericReparseBuffer (variable): The data in this variable buffer takes the following format.

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
Type																															
...																															
DataBuffer (variable)																															
...																															

Type (8 bytes): A 64-bit unsigned integer value describing the type and format of the data stored in the **DataBuffer** field. The valid values for this field are:

Value	Meaning
NFS_SPECFILE_LNK 0x00000000014B4E4C	Indicates that the **DataBuffer** field has a Unicode string containing the symbolic link data.
NFS_SPECFILE_CHR 0x0000000000524843	Indicates that the **DataBuffer** field has two 32–bit integers that contain the major and minor device numbers for the character special device created by the Network File System client.

Value	Meaning
NFS_SPECFILE_BLK 0x00000000004B4C42	Indicates that the **DataBuffer** field has two 32–bit integers that contain the major and minor device numbers for the block special device created by the Network File System client.
NFS_SPECFILE_FIFO 0x000000004F464946	Indicates that the file containing the NFS reparse point is a named pipe device created by the Network File System client. The **DataBuffer** field is empty.
NFS_SPECFILE_SOCK 0x000000004B434F53	Indicates that the file containing the NFS reparse point is a socket device created by the Network File System client. The **DataBuffer** field is empty.

DataBuffer (variable): A variable buffer that has the following formats depending upon the **Type** field defined earlier.

- **NFS_SPECFILE_CHR** and **NFS_SPECFILE_BLK**: The **DataBuffer** field contains two 32-bit integers that represent major and minor device numbers.
- **NFS_SPECFILE_LNK**: The **DataBuffer** field contains the symbolic link target path specified by the Network File System client in its NFSPROC_SYMLINK request, [RFC1813] section 3.3.10 and [RFC1094] section 2.2.14, represented in Unicode format and not NULL-terminated. The upper limit on the size of the symbolic link data is 2050 bytes.
- **NFS_SPECFILE_FIFO** and **NFS_SPECFILE_SOCK**: The **DataBuffer** field is empty.

2.1.2.7 LX SYMLINK REPARSE_DATA_BUFFER

The **LX SYMLINK Reparse Data Buffer** data element is a subtype of section REPARSE_DATA_BUFFER, which contains information about symbolic files generated by WSL (Windows Subsystem for Linux).

Expand table

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
ReparseTag																															
ReparseDataLength																Reserved															
Version																															
Target (variable)																															
...																															

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

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

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

Version (4 bytes): A 32-bit field. This field defines the layout of the **Target** field. This field MUST be set to 2.

Target (variable): An array of 8-byte characters that contains the target path of the symlink.

Last updated on 11/21/2025

2.1.3 FILE_OBJECTID_BUFFER Structure

Article02/14/2019

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

- FILE_OBJECTID_BUFFER Type 1
- FILE_OBJECTID_BUFFER Type 2

2.1.3.1 FILE_OBJECTID_BUFFER Type 1

06/10/2025

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

Expand table

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 (16 bytes)																															
...																															
...																															
BirthVolumeId (16 bytes)																															
...																															
...																															
BirthObjectId (16 bytes)																															
...																															
...																															
DomainId (16 bytes)																															
...																															
...																															

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

BirthVolumeId (16 bytes): A 16-byte GUID that uniquely identifies the volume on which the object resided when the object identifier was created, or zero if the volume had no object identifier at that time. After copy operations, move operations, or other file operations, this value is potentially different from the object identifier of the volume on which the object presently resides.

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

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

2.1.3.2 FILE_OBJECTID_BUFFER Type 2

Article04/07/2025

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

Expand table

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 (16 bytes)																															
...																															
...																															
ExtendedInfo (48 bytes)																															
...																															
...																															

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

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

2.1.4 Alternate Data Streams

Article04/07/2025

A file system MAY<8> support alternate data streams within a file or a directory. For a general description of file streams, section 1.1.

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

For more information on stream naming, see section 2.1.5; for more information on streams in general, see section 5.

2.1.5 Pathname

Article04/07/2025

A pathname has the following characteristics:

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

2.1.5.1 Dot Directory Names

Article02/10/2025

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

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

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

Some examples to illustrate:

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

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

2.1.5.2 Filename

Article04/07/2025

- All Unicode characters are legal in a filename except the following:
  - The characters

    ```
    " \ / : | < > * ?
    ```

  - Control characters, ranging from 0x00 through 0x1F.
- A filename MUST be at least one character but no more than 255 characters in length.

2.1.5.2.1 8.3 Filename

Article04/07/2025

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

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

2.1.5.3 Streamname

06/10/2025

- All Unicode characters are legal in a streamname component except the following:
  - The characters \ / :
  - Control character 0x00.
  - A streamname MUST be no more than 255 characters in length.
- A zero-length streamname denotes the default stream.

See section 5 for additional information on alternate streams in the NTFS file system.

2.1.5.4 Streamtype

Article04/07/2025

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

2.1.6 Share name

Article02/14/2019

A share name has the following characteristics:

- A share name MUST be no more than 80 characters in length.
- The following characters are illegal in a share name:

```
" \ / [ ] : | < > + = ; , * ?
```

- Control characters in range 0x00 through 0x1F, inclusive, are illegal in a share name.
- All other Unicode characters are legal.

2.1.7 FILE_NAME_INFORMATION

Article04/07/2025

The **FILE_NAME_INFORMATION** data element is as follows.

Expand table

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 specifies the length, in bytes, of the file name contained within the **FileName** field.

FileName (variable): A sequence of Unicode characters containing a pathname (section 2.1.5). The meaning of the pathname depends on the operation. The name string is not null-terminated. There are scenarios where one or more padding characters can be at the end of the string due to buffer alignment requirements, but their presence and their values MUST NOT be relied upon. When working with this field, use **FileNameLength** to determine the length of the file name rather than assuming the presence of a trailing null delimiter.

2.1.8 Boolean

06/10/2025

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

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

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

2.1.9 64-bit file ID

Article09/20/2023

A **64-bit file ID** value uniquely identifies a file within a given volume. This identifier is generated and stored by the file system. The identifier SHOULD<10> be unique to the volume and stable until the file is deleted.

For file systems that do not support a 64-bit file ID, this field MUST be set to 0, and MUST be ignored.

For files for which a unique 64-bit file ID cannot be established, this field MUST be set to 0xFFFFFFFFFFFFFFFF, and MUST be ignored.

2.1.10 128-bit file ID

Article09/20/2023

A **128-bit file ID** value uniquely identifies a file within a given volume. This identifier is generated and stored by the file system. The identifier SHOULD<11> be unique to the volume and stable until the file is deleted.

For file systems that do not support a **128-bit file ID**, this field MUST be set to 0, and MUST be ignored.

For files for which a unique **128-bit file ID** cannot be established, this field MUST be set to 0xFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF, and MUST be ignored.

2.1.11 STORAGE_OFFLOAD_TOKEN

Article04/07/2025

The **STORAGE_OFFLOAD_TOKEN** structure contains the Token to be used as a representation of the data contained within the portion of the file specified in the FSCTL_OFFLOAD_READ_INPUT data element at the time of the FSCTL_OFFLOAD_READ operation. This Token is used in FSCTL_OFFLOAD_READ and FSCTL_OFFLOAD_WRITE operations. The format of the data within this field is either vendor-specific or of a well-known type. The contents of this field MUST NOT be modified during subsequent operations.<12>

The **TokenType** and **TokenIdLength** fields of **STORAGE_OFFLOAD_TOKEN** structure MUST be sent in big-endian format. The **TokenId** field is a stream of bytes and has no endian property.

The **STORAGE_OFFLOAD_TOKEN** structure is as follows.

Expand table

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
TokenType																															
Reserved																TokenIdLength															
TokenId (504 bytes)																															
...																															
...																															

TokenType (4 bytes): A 32-bit unsigned integer that defines the type of Token that is contained within the **STORAGE_OFFLOAD_TOKEN** structure. This field MUST contain one of the following values.

Expand table

Value	Meaning
STORAGE_OFFLOAD_TOKEN_TYPE_ZERO_DATA 0xFFFF0001	A well-known Token that indicates that the data logically represented by the Token is logically equivalent to zero.<13>
Reserved 0xFFFF0002 – 0xFFFFFFFF	Reserved for other well-known Tokens currently undefined.
Any other value.	A vendor-specific Token format is contained within the **Token** field.

Reserved (2 bytes): A 16-bit unsigned integer that is reserved. This field SHOULD be set to 0x0000 and MUST be ignored.

TokenIdLength (2 bytes): A 16-bit unsigned integer that defines the length of the **TokenId** field in bytes.

TokenId (504 bytes): A 504-byte unsigned integer that contains opaque vendor-specific data.

2.2 Status Codes

Article04/27/2022

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 lists the common error codes that are directly generated by the function. Error codes can also be generated by code below the file system (such as RAID drivers or disk drivers) or above the file system (such as virus scanners).

A server SHOULD return a status of STATUS_INVALID_DEVICE_REQUEST when a message is not supported remotely or is not supported on the file system on which the file or directory handle specified exists.<14><15>

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

2.3 FSCTL Structures

Article12/14/2021

A process invokes an FSCTL on a handle to perform an action against the file or directory associated with the handle. When a server receives an FSCTL request, it SHOULD use the information in the request, which includes a handle and, optionally, an input data buffer, to perform the requested action. How a server performs the action requested by an FSCTL is implementation-dependent. <16>

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

FSCTL name	FSCTL function number
FSCTL_CREATE_OR_GET_OBJECT_ID	0X900C0
FSCTL_DELETE_OBJECT_ID	0X900A0
FSCTL_DELETE_REPARSE_POINT	0X900AC
FSCTL_DUPLICATE_EXTENTS_TO_FILE	0X98344
FSCTL_DUPLICATE_EXTENTS_TO_FILE_EX	0x983E8
FSCTL_FILESYSTEM_GET_STATISTICS	0X90060
FSCTL_FILE_LEVEL_TRIM	0X98208
FSCTL_FIND_FILES_BY_SID	0X9008F
FSCTL_GET_COMPRESSION	0X9003C
FSCTL_GET_INTEGRITY_INFORMATION	0X9027C
FSCTL_GET_NTFS_VOLUME_DATA	0X90064
FSCTL_GET_REFS_VOLUME_DATA	0X902D8
FSCTL_GET_OBJECT_ID	0X9009C
FSCTL_GET_REPARSE_POINT	0X900A8

FSCTL name	FSCTL function number
FSCTL_GET_RETRIEVAL_POINTER_COUNT	0x9042B
FSCTL_GET_RETRIEVAL_POINTERS	0X90073
FSCTL_GET_RETRIEVAL_POINTERS_AND_REFCOUNT	0x903D3
FSCTL_IS_PATHNAME_VALID	0X9002C
FSCTL_LMR_SET_LINK_TRACKING_INFORMATION	0X1400EC
FSCTL_MARK_HANDLE	0x900FC
FSCTL_OFFLOAD_READ	0X94264
FSCTL_OFFLOAD_WRITE	0X98268
FSCTL_PIPE_PEEK	0X11400C
FSCTL_PIPE_TRANSCEIVE	0X11C017
FSCTL_PIPE_WAIT	0X110018
FSCTL_QUERY_ALLOCATED_RANGES	0X940CF
FSCTL_QUERY_FAT_BPB	0X90058
FSCTL_QUERY_FILE_REGIONS	0X90284
FSCTL_QUERY_ON_DISK_VOLUME_INFO	0X9013C
FSCTL_QUERY_SPARING_INFO	0X90138
FSCTL_READ_FILE_USN_DATA	0X900EB
FSCTL_RECALL_FILE	0X90117
FSCTL_REFS_STREAM_SNAPSHOT_MANAGEMENT	0x90440
FSCTL_SET_COMPRESSION	0X9C040
FSCTL_SET_DEFECT_MANAGEMENT	0X98134
FSCTL_SET_ENCRYPTION	0X900D7
FSCTL_SET_INTEGRITY_INFORMATION	0X9C280
FSCTL_SET_INTEGRITY_INFORMATION_EX	0x90380
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_SET_ZERO_ON_DEALLOCATION	0X90194
FSCTL_SIS_COPYFILE	0X90100
FSCTL_WRITE_USN_CLOSE_RECORD	0X900EF

2.3.1 FSCTL_CREATE_OR_GET_OBJECT_ID Request

Article02/14/2019

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

This message does not contain any additional data elements.

2.3.2 FSCTL_CREATE_OR_GET_OBJECT_ID Reply

Article04/07/2025

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

The buffer can be either Type 1 or Type 2 as follows:

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

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

This message also returns a status code as specified in section 2.2. Upon success, the status code returned by the function that processes this FSCTL is STATUS_SUCCESS. The most common error codes are listed in the following table.

Expand table

Error code	Meaning
STATUS_DUPLICATE_NAME 0xC00000BD	The file has no object ID yet, and the file system is unable to generate a unique ID (to this volume).<17>
STATUS_INVALID_PARAMETER 0xC000000D	The handle is not to a file or directory, or the output buffer is not large enough to contain a FILE_OBJECTID_BUFFER structure.
STATUS_MEDIA_WRITE_PROTECTED 0xC00000A2	The volume is write-protected and changes to it cannot be made. This error code is returned even if the file already has an object ID assigned to it.
STATUS_INVALID_DEVICE_REQUEST 0xC0000010	The file system does not support the use of object IDs.

2.3.3 FSCTL_DELETE_OBJECT_ID Request

Article04/07/2025

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

This message does not contain any additional data elements.

2.3.4 FSCTL_DELETE_OBJECT_ID Reply

Article04/07/2025

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 section 2.2. Upon success, the status code returned by the function that processes this FSCTL is STATUS_SUCCESS. The most common error codes are listed in the following table.

Expand table

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

2.3.5 FSCTL_DELETE_REPARSE_POINT Request

Article04/07/2025

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 (including subtypes). Both the REPARSE_GUID_DATA_BUFFER and the REPARSE_DATA_BUFFER structures begin with a **ReparseTag** field. The **ReparseTag** value uniquely identifies the filter driver that creates/uses the reparse point, and the application's filter driver processes the reparse point data as either a REPARSE_GUID_DATA_BUFFER or a REPARSE_DATA_BUFFER, depending on the structure implemented by the filter driver for that type of reparse point.

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

2.3.6 FSCTL_DELETE_REPARSE_POINT Reply

Article04/07/2025

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 section 2.2. Upon success, the status code returned by the function that processes this FSCTL is STATUS_SUCCESS. The most common error codes are listed in the following table.

Expand table

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

2.3.7 FSCTL_DUPLICATE_EXTENTS_TO_FILE Request

Article09/26/2024

The FSCTL_DUPLICATE_EXTENTS_TO_FILE<18> request message requests that the server copy the specified portion of one file (that is the source file) into a specified portion of another file (target file) on the same volume. The logical sizes of the portions have to be the same. The two files involved in this operation can refer to the same file, but in that case, the logical portions have to refer to disjoint regions on the file. The FSCTL is sent on a handle opened to the target file.

When used locally, the request message takes the form of DUPLICATE_EXTENTS_DATA as specified in section 2.3.7.1. When used remotely with [MS-SMB2], the request message takes the form of SMB2_DUPLICATE_EXTENTS_DATA as specified in section 2.3.7.2.

2.3.7.1 DUPLICATE_EXTENTS_DATA

A **DUPLICATE_EXTENTS_DATA** data element is defined as follows:

Expand table

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
FileHandle																															
...																															
SourceFileOffset																															
...																															
TargetFileOffset																															
...																															
ByteCount																															
...																															

FileHandle (8 bytes): A HANDLE ([MS-DTYP] section 2.2.16) data type that is an identifier of the open to the source file.

SourceFileOffset (8 bytes): A 64-bit signed integer that contains the file offset, in bytes, of the start of a range of bytes in a source file from which the data is to be copied. The value of this field MUST be greater than or equal to 0x0000000000000000 and MUST be aligned to a logical cluster boundary.

TargetFileOffset (8 bytes): A 64-bit signed integer that contains the file offset, in bytes, of the start of a range of bytes in a target file to which the data is to be copied. The value of this field MUST be greater than or equal to 0x0000000000000000 and MUST be aligned to a logical cluster boundary.

ByteCount (8 bytes): A 64-bit signed integer that contains the number of bytes to copy from source to target. The value of this field MUST be greater than or equal to 0x0000000000000000 and MUST be aligned to a logical cluster boundary.

Last updated on 11/21/2025

2.3.7.2 SMB2_DUPLICATE_EXTENTS_DATA

Article02/14/2019

A SMB2_DUPLICATE_EXTENTS_DATA data element is defined as follows:

0	1	2	3	4	5	6	7	8	9	10	1	2	3	4	5	6	7	8	9	20	1	2	3	4	5	6	7	8	9	30	1
SourceFileID																															
...																															
...																															
...																															
SourceFileOffset																															
...																															
TargetFileOffset																															
...																															
ByteCount																															
...																															

SourceFileID (16 bytes): An SMB2_FILEID structure, as specified in [MS-SMB2] section 2.2.14.1, that is an identifier of the open to the source file.

SourceFileOffset (8 bytes): A 64-bit signed integer that contains the file offset, in bytes, of the start of a range of bytes in a source file from which the data is to be copied. The value of this field MUST be greater than or equal to 0x0000000000000000 and MUST be aligned to a logical cluster boundary.

TargetFileOffset (8 bytes): A 64-bit signed integer that contains the file offset, in bytes, of the start of a range of bytes in a target file to which the data is to be copied. The value of this field MUST be greater than or equal to 0x0000000000000000 and MUST be aligned to a logical cluster boundary.

ByteCount (8 bytes): A 64-bit signed integer that contains the number of bytes to copy from source to target. The value of this field MUST be greater than or equal to 0x0000000000000000 and MUST be aligned to a logical cluster boundary.

2.3.8 FSCTL_DUPLICATE_EXTENTS_TO_FILE Reply

Article04/07/2025

This message returns the result of the FSCTL_DUPLICATE_EXTENTS_TO_FILE<19> request.

The only data item this message returns is a status code, as specified in section 2.2. Upon success, the status code returned by the function that processes this FSCTL SHOULD<20> be STATUS_SUCCESS. The most common error codes are listed in the following table.

Expand table

Error Code	Meaning
STATUS_NOT_SUPPORTED 0xC00000BB	• The source and target destination ranges overlap on the same file. • Source file is sparse, while target is a non-sparse file. • The source range is beyond the source file's allocation size.
STATUS_INVALID_PARAMETER 0xC000000D	The FileHandle parameter is either invalid or does not represent a handle to an opened file on the same volume.
STATUS_INSUFFICIENT_RESOURCES 0xC000009A	There were insufficient resources to complete the operation.
STATUS_DISK_FULL 0xC000007F	The disk is full.
STATUS_MEDIA_WRITE_PROTECTED 0xC00000A2	The volume is read-only.
STATUS_INVALID_DEVICE_REQUEST 0xC0000010	The file system does not support duplicating extents.

2.3.9 FSCTL_DUPLICATE_EXTENTS_TO_FILE_EX Request

Article02/10/2025

The FSCTL_DUPLICATE_EXTENTS_TO_FILE_EX<21> request message requests that the server copy the specified portion of the source file into a specified portion of the target file on the same volume. The logical sizes of the portions MUST be the same. The two files involved in this operation can refer to the same file but the logical portions have to refer to disjoint regions on the file. The FSCTL is sent on a handle opened to the target file. When the DUPLICATE_EXTENTS_DATA_EX_SOURCE_ATOMIC flag isn't set, the behavior is identical to FSCTL_DUPLICATE_EXTENTS_TO_FILE. When the flag is set, duplication is atomic from the source's point of view. It means duplication fully succeeds or fails without side effect (when only part of source file region is duplicated).

When used locally, the request message takes the form of DUPLICATE_EXTENTS_DATA_EX as specified in section 2.3.9.1. When used remotely with [MS-SMB2], the request message takes the form of SMB2_DUPLICATE_EXTENTS_DATA_EX as specified in section 2.3.9.2.

2.3.9.1 DUPLICATE_EXTENTS_DATA_EX

Article05/28/2021

A DUPLICATE_EXTENTS_DATA_EX data element is defined as follows:

0	1	2	3	4	5	6	7	8	9	10	1	2	3	4	5	6	7	8	9	20	1	2	3	4	5	6	7	8	9	30	1
StructureSize																															
...																															
FileHandle																															
...																															
SourceFileOffset																															
...																															
TargetFileOffset																															
...																															
ByteCount																															
...																															
Flags																															

StructureSize (8 bytes): A SIZE_T [MS-DTYP] section 2.2.43) data type that specifies the size of the structure, in bytes.

FileHandle (8 bytes): A HANDLE ([MS-DTYP] section 2.2.16) data type that is an identifier of the open to the source file.

SourceFileOffset (8 bytes): A 64-bit signed integer that contains the file offset, in bytes, of the start of a range of bytes in a source file from which the data is to be copied. The value of this field MUST be greater than or equal to 0x0000000000000000 and MUST be aligned to a logical cluster boundary.

TargetFileOffset (8 bytes): A 64-bit signed integer that contains the file offset, in bytes, of the start of a range of bytes in a target file to which the data is to be copied. The value of this field MUST be greater than or equal to 0x0000000000000000 and MUST be aligned to a logical cluster boundary.

ByteCount (8 bytes): A 64-bit signed integer that contains the number of bytes to copy from source to target. The value of this field MUST be greater than or equal to 0x0000000000000000 and MUST be aligned to a logical cluster boundary.

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

Value	Meaning

Value	Meaning
DUPLICATE_EXTENTS_DATA_EX_SOURCE_ATOMIC 0x00000001	Indicates that duplication is atomic from source point of view.

2.3.9.2 SMB2_DUPLICATE_EXTENTS_DATA_EX

Article09/20/2023

A **SMB2_DUPLICATE_EXTENTS_DATA_EX** data element is defined as follows:

0	1	2	3	4	5	6	7	8	9	10	1	2	3	4	5	6	7	8	9	20	1	2	3	4	5	6	7	8	9	30	1
StructureSize																															
...																															
SourceFileID																															
...																															
...																															
...																															
SourceFileOffset																															
...																															
TargetFileOffset																															
...																															
ByteCount																															
...																															
Flags																															
Reserved																															

StructureSize (8 bytes): A 64-bit unsigned integer value that specifies the size of the structure, in bytes. This field MUST be set to 0x30.

SourceFileID (16 bytes): An SMB2_FILEID structure, as specified in [MS-SMB2] section 2.2.14.1, that is an identifier of the open to the source file.

SourceFileOffset (8 bytes): A 64-bit signed integer that contains the file offset, in bytes, of the start of a range of bytes in a source file from which the data is to be copied. The value of this field MUST be greater than or equal to 0x0000000000000000 and MUST be aligned to a logical cluster boundary.

TargetFileOffset (8 bytes): A 64-bit signed integer that contains the file offset, in bytes, of the start of a range of bytes in a target file to which the data is to be copied. The value of this field MUST be greater than or equal to 0x0000000000000000 and MUST be aligned to a logical cluster boundary.

ByteCount (8 bytes): A 64-bit signed integer that contains the number of bytes to copy from source to target. The value of this field MUST be greater than or equal to 0x0000000000000000 and MUST be aligned to a logical cluster boundary.

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

Value	Meaning
DUPLICATE_EXTENTS_DATA_EX_SOURCE_ATOMIC 0x00000001	Indicates that duplication is atomic from source point of view.

Reserved (4 bytes): This field SHOULD be set to zero and MUST be ignored.

2.3.10 FSCTL_DUPLICATE_EXTENTS_TO_FILE_EX Reply

Article04/07/2025

This message returns the result of the FSCTL_DUPLICATE_EXTENTS_TO_FILE_EX request<22>.

The only data item this message returns is a status code, as specified in section 2.2. Upon success, the status code returned by the function that processes this FSCTL SHOULD be STATUS_SUCCESS. The most common error codes are listed in the following table.

Expand table

Error Code	Meaning
STATUS_NOT_SUPPORTED 0xC00000BB	• The source and target destination ranges overlap on the same file. • Source file is sparse, while target is a non-sparse file. • The source range is beyond the source file's allocation size.
STATUS_INVALID_PARAMETER 0xC000000D	The FileHandle parameter is either invalid or does not represent a handle to an opened file on the same volume.
STATUS_INSUFFICIENT_RESOURCES 0xC000009A	There were insufficient resources to complete the operation.
STATUS_DISK_FULL 0xC000007F	The disk is full.
STATUS_MEDIA_WRITE_PROTECTED 0xC00000A2	The volume is read-only.
STATUS_INVALID_DEVICE_REQUEST 0xC0000010	The file system does not support duplicating extents.

2.3.11 FSCTL_FILESYSTEM_GET_STATISTICS Request

Article04/07/2025

This message requests that the server return the statistical information of the file system such as Type, Version, and so on, as specified in FSCTL_FILESYSTEM_GET_STATISTICS reply, for the file or directory associated with the handle on which this FSCTL was invoked.<23>

This message does not contain any additional data elements.

2.3.12 FSCTL_FILESYSTEM_GET_STATISTICS Reply

Article04/07/2025

This message returns the result of the FSCTL_FILESYSTEM_GET_STATISTICS request message as a pair of structures: a generic structure, FILESYSTEM_STATISTICS, optionally followed by a file system type specific structure that can be either NTFS_STATISTICS, FAT_STATISTICS, or EXFAT_STATISTICS, depending on the underlying file system type. There is one pair of these structures for each processor.<24>

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

The statistics structures contain fields that can overflow during the server's lifetime. This is by design. When an overflow occurs, the value just wraps. For example, 0XFFFFF000 + 0x2000 will result in 0x1000.

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

This message also returns a status code as specified in section 2.2. Upon success, the status code returned by the function that processes this FSCTL is STATUS_SUCCESS. The most common error codes are listed in the following table.

Expand table

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

2.3.12.1 FILESYSTEM_STATISTICS

Article04/27/2022

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

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
FileSystemType																Version															
SizeOfCompleteStructure																															
UserFileReads																															
UserFileReadBytes																															
UserDiskReads																															
UserFileWrites																															
UserFileWriteBytes																															
UserDiskWrites																															
MetaDataReads																															
MetaDataReadBytes																															
MetaDataDiskReads																															
MetaDataWrites																															
MetaDataWriteBytes																															
MetaDataDiskWrites																															

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

Value	Meaning
FILESYSTEM_STATISTICS_TYPE_NTFS 0x0001	The file system is an NTFS file system. If this value is set, this structure is followed by an NTFS_STATISTICS structure.
FILESYSTEM_STATISTICS_TYPE_FAT 0x0002	The file system is a FAT file system. If this value is set, this structure is followed by a FAT_STATISTICS structure.
FILESYSTEM_STATISTICS_TYPE_EXFAT 0x0003	The file system is an exFAT file system. If this value is set, this structure is followed by an EXFAT_STATISTICS structure.
FILESYSTEM_STATISTICS_TYPE_REFS 0x0004	The file system is an ReFS file system. If this value is set, this structure is not followed by a structure specific to file system type.

Version (2 bytes): A 16-bit unsigned integer value containing the version. This field MUST be set to the value 0x0001.

SizeOfCompleteStructure (4 bytes): A 32-bit unsigned integer value that indicates the size, in bytes, of this structure plus the size of the file system-specific structure that follows this structure, each rounded up to a multiple of 64, then the sum is multiplied by the number of processors. For example, if the size of FILESYSTEM_STATISTICS is 0x38, the size of NTFS_STATISTICS is 0XD4, and there are two processors, the size of the buffer allocated is 0x280. This is the sum of the sizes of the NTFS_STATISTICS structure and the FILESYSTEM_STATISTICS structure, both rounded up to a multiple of 64 (0x40 + 0x100 = 0x140), and multiplied by the number of processors.

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

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

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

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

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

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

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

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

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

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

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

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

2.3.12.2 NTFS_STATISTICS

Article04/27/2022

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

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
LogFileFullExceptions																															
OtherExceptions																															
MftReads																															
MftReadBytes																															
MftWrites																															
MftWriteBytes																															
MftWritesUserLevel																															
...																															
MftWritesFlushForLogFileFull																				MftWritesLazyWriter											
MftWritesUserRequest																				Padding1											
Mft2Writes																															
Mft2WriteBytes																															
Mft2WritesUserLevel																															
...																															
Mft2WritesFlushForLogFileFull																				Mft2WritesLazyWriter											
Mft2WritesUserRequest																				Padding2											
RootIndexReads																															
RootIndexReadBytes																															
RootIndexWrites																															
RootIndexWriteBytes																															
BitmapReads																															
BitmapReadBytes																															
BitmapWrites																															
BitmapWriteBytes																															
BitmapWritesFlushForLogFileFull																				BitmapWritesLazyWriter											
BitmapWritesUserRequest																				BitmapWritesUserLevel											

...	
MftBitmapReads	
MftBitmapReadBytes	
MftBitmapWrites	
MftBitmapWriteBytes	
MftBitmapWritesFlushForLogFileFull	MftBitmapWritesLazyWriter
MftBitmapWritesUserRequest	MftBitmapWritesUserLevel
...	
...	Padding3
UserIndexReads	
UserIndexReadBytes	
UserIndexWrites	
UserIndexWriteBytes	
LogFileReads	
LogFileReadBytes	
LogFileWrites	
LogFileWriteBytes	
Allocate (40 bytes)	
...	
...	

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

2.3.12.2.1 MftWritesUserLevel

06/10/2025

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

The **MftWritesUserLevel** structure is as follows.

Expand table

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

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

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

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

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

2.3.12.2.2 Mft2WritesUserLevel

Article07/03/2024

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

The **Mft2WritesUserLevel** structure is as follows.

Expand table

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

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

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

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

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

2.3.12.2.3 BitmapWritesUserLevel

Article09/20/2023

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

The **BitmapWritesUserLevel** structure is as follows.

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

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

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

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

2.3.12.2.4 MftBitmapWritesUserLevel

Article09/20/2023

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

The **MftBitmapWritesUserLevel** structure is as follows.

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

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

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

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

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

2.3.12.2.5 Allocate

Article02/10/2025

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

The **Allocate** structure is as follows.

Expand table

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

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

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

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

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

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

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

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

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

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

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

2.3.12.3 FAT_STATISTICS

Article04/07/2025

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

The **FAT_STATISTICS** data element is as follows:

Expand table

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

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

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

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

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

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

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

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

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

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

2.3.12.4 EXFAT_STATISTICS

Article12/14/2021

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

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

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

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

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

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

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

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

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

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

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

2.3.13 FSCTL_FILE_LEVEL_TRIM Request

Article08/24/2020

The FSCTL_FILE_LEVEL_TRIM operation informs the underlying storage medium that the contents of the given range of the file no longer needs to be maintained. This message allows the storage medium to manage its space more efficiently. This operation is required most commonly for Solid State Devices (SSD), as well as for thinly provisioned storage environments.

The **FILE_LEVEL_TRIM** 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
Key																															
NumRanges																															
Ranges (variable)																															
...																															

Key (4 bytes): This field is used for byte range locks to uniquely identify different consumers of byte range locks on the same thread. Typically, this field is used only by remote protocols such as SMB or SMB2.

NumRanges (4 bytes): A count of how many **Offset**, **Length** pairs follow in the data item.

Ranges (variable): An array of zero or more FILE_LEVEL_TRIM_RANGE (section 2.3.13.1) data elements. The **NumRanges** field contains the number of **FILE_LEVEL_TRIM_RANGE** data elements in the array.

2.3.13.1 FILE_LEVEL_TRIM_RANGE

Article08/24/2020

The **FILE_LEVEL_TRIM_RANGE** 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
Offset																															
...																															
Length																															
...																															

Offset (8 bytes): A 64-bit unsigned integer that contains a byte offset into the given file at which to start the trim request.

Length (8 bytes): A 64-bit unsigned integer that contains the length, in bytes, of how much of the file to trim, starting at **Offset**.

2.3.14 FSCTL_FILE_LEVEL_TRIM Reply

Article04/07/2025

This message returns the results of the FSCTL_FILE_LEVEL_TRIM Request (section 2.3.13).

The **FILE_LEVEL_TRIM_OUTPUT** data element follows.

Expand table

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
NumRangesProcessed																															

NumRangesProcessed (4 bytes): A 32-bit unsigned integer identifying the number of input ranges that were processed.

This message returns a status code as specified in section 2.2. Upon success, the status code returned by the function that processes this FSCTL is STATUS_SUCCESS. The most common error codes are listed in the following table.

Expand table

Error code	Meaning
STATUS_INVALID_PARAMETER 0xC000000D	The given file is compressed or encrypted, or the size of the input buffer is smaller than the size of the **FILE_LEVEL_TRIM** data element, or no FILE_LEVEL_TRIM_RANGE (section 2.3.13.1) structures were given, or the output buffer is smaller than the size of **FILE_LEVEL_TRIM_OUTPUT**.
STATUS_INVALID_DEVICE_REQUEST 0xC0000010	The file system does not support this operation.
STATUS_INTEGER_OVERFLOW 0xC0000095	An operation on a parameter in the FSCTL_FILE_LEVEL_TRIM input structure overflowed 64 bits.
STATUS_NO_RANGES_PROCESSED 0xC0000460	The operation was successful, but no range was processed.

2.3.15 FSCTL_FIND_FILES_BY_SID Request

Article04/07/2025

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

The FIND_BY_SID_DATA data element is as follows.

Expand table

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

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

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

2.3.16 FSCTL_FIND_FILES_BY_SID Reply

Article04/07/2025

The FSCTL_FIND_FILES_BY_SID Reply message returns the results of the FSCTL_FIND_FILES_BY_SID Request (section 2.3.15) as an array of FILE_NAME_INFORMATION (section 2.1.7) data elements containing relative pathnames (section 2.1.5), one for each matching file or directory that is found, in no necessary order. All returned file names MUST be relative to the directory on which the FSCTL_FIND_FILES_BY_SID Request was issued. This returns as many **FILE_NAME_INFORMATION** data elements as will fit in the provided output buffer. The beginning of each **FILE_NAME_INFORMATION** data element MUST be aligned to an 8-byte boundary, as measured from the beginning of the buffer. The last **FILE_NAME_INFORMATION** structure returned MAY<25> contain trailing padding.

This message also returns a status code as specified in section 2.2. Upon success, the status code returned by the function that processes this FSCTL is STATUS_SUCCESS. The most common error codes are listed in the following table.

Expand table

Status code	Meaning
STATUS_NO_QUOTAS_FOR_ACCOUNT 0x0000010D	Quota tracking is not enabled; therefore, the file system does not keep a record of file owners. This is considered a success code. The reply MUST NOT contain any data elements.
STATUS_INVALID_PARAMETER 0xC000000D	The handle specified is not the handle to a directory.
STATUS_ACCESS_DENIED 0xC0000022	Neither the SeManageVolumePrivilege nor the SeBackupPrivilege, as specified in [MS-LSAD] section 3.1.1.2.1, privilege is held.
STATUS_BUFFER_TOO_SMALL 0xC0000023	The output buffer is not large enough to contain the **FILE_NAME_INFORMATION** structure (including any trailing padding) for the first matching file or directory.
STATUS_INVALID_USER_BUFFER 0xC00000E8	The input buffer is less than the size of a long integer (4 bytes) plus the length of the SID provided, or the input or output buffer is not aligned to the native word size of the platform, or the size of the output buffer is less than the minimum size of a **FILE_NAME_INFORMATION** structure (8 bytes), or the restart value is greater than 1.

When the status code is STATUS_SUCCESS, the responder MUST retain an implementation-dependent indication of where the directory processing ended, which is required to support a

subsequent FSCTL_FIND_FILES_BY_SID Request with the **Restart** field set to 0x00000000. For an example of FSCTL_FIND_FILES_BY_SID restart handling, see [MS-FSA] section 2.1.5.10.8.

2.3.17 FSCTL_GET_COMPRESSION Request

Article04/07/2025

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

This message does not contain any additional data elements.

2.3.18 FSCTL_GET_COMPRESSION Reply

Article04/07/2025

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.

Expand table

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

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

Expand table

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

The actual file or directory compression format is implementation-dependent. <26>

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

This message also returns a status code as specified in section 2.2. Upon success, the status code returned by the function that processes this FSCTL is STATUS_SUCCESS. The most common error codes are listed in the following table.

Expand table

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. <27>

2.3.19 FSCTL_GET_INTEGRITY_INFORMATION Request

Article04/07/2025

The FSCTL_GET_INTEGRITY_INFORMATION Request message requests that the server return the current integrity state of the file or directory associated with the handle on which this FSCTL is invoked.<28>

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

This message does not contain additional data elements.

2.3.20 FSCTL_GET_INTEGRITY_INFORMATION Reply

Article04/23/2024

The FSCTL_GET_INTEGRITY_INFORMATION Reply message returns the results of the FSCTL_GET_INTEGRITY_INFORMATION Request (section 2.3.19) and indicates the current integrity state of the file or directory.

The **FSCTL_GET_INTEGRITY_INFORMATION_BUFFER** data element is as follows.

Expand table

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
ChecksumAlgorithm																Reserved															
Flags																															
ChecksumChunkSizeInBytes																															
ClusterSizeInBytes																															

ChecksumAlgorithm (2 bytes): For **ReFS v1**, the field MUST be set to one of the following standard values.

Expand table

Value	Meaning
CHECKSUM_TYPE_NONE 0x0000	The file or directory is not configured to use integrity.
CHECKSUM_TYPE_CRC64 0x0002	The file or directory is configured to use a CRC64 checksum to provide integrity.
All other values	Reserved for future use and MUST NOT be used.

For **ReFS v2**, the field MUST be set to one of the following standard values.

Expand table

Value	Meaning
CHECKSUM_TYPE_NONE 0x0000	The file or directory is not configured to use integrity.
CHECKSUM_TYPE_CRC32 0x0001	The file or directory is configured to use a CRC32 checksum to provide integrity.

Value	Meaning
CHECKSUM_TYPE_CRC64 0x0002	The file or directory is configured to use a CRC64 checksum to provide integrity.
All other values	Reserved for future use and MUST NOT be used.

Reserved (2 bytes): A 16-bit reserved value. This field MUST be set to 0x0000 and MUST be ignored.

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

Expand table

Value	Meaning
FSCTL_INTEGRITY_FLAG_CHECKSUM_ENFORCEMENT_OFF 0x00000001	Indicates that checksum enforcement is not currently enabled on the target file.
All other values	Reserved for future use and MUST NOT be used.

ChecksumChunkSizeInBytes (4 bytes): A 32-bit unsigned integer specifying the size in bytes of each chunk in a stream that is configured with integrity.

ClusterSizeInBytes (4 bytes): A 32-bit unsigned integer specifying the size of a cluster for this volume in bytes.

This message also returns a status code, as specified in section 2.2. Upon success, the status code returned by the function that processes this FSCTL MUST be STATUS_SUCCESS or one of the following.

Expand table

Error code	Meaning
STATUS_INVALID_PARAMETER 0xC000000D	The output buffer length is less than the size of the FSCTL_GET_INTEGRITY_INFORMATION_BUFFER data element, or the handle is not to a file or directory.
STATUS_INVALID_DEVICE_REQUEST 0xC0000010	The volume does not support integrity.

2.3.21 FSCTL_GET_NTFS_VOLUME_DATA Request

Article04/07/2025

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

This message does not contain any parameters.

2.3.22 FSCTL_GET_NTFS_VOLUME_DATA Reply

Article04/07/2025

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

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

Expand table

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

MftZoneEnd
...

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

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

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

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

TotalReserved (8 bytes): A 64-bit signed integer that contains the number of reserved clusters in the specified volume. Reserved clusters are free clusters reserved for when the volume becomes full. Reserved clusters used to guarantee clusters are available at points when the file system can't properly report allocation failures.

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

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

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

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

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

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

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

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

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

This message also returns a status code as specified in section 2.2. Upon success, the status code returned directly by the function that processes this FSCTL is STATUS_SUCCESS. The most common error codes are listed in the following table.

Expand table

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

2.3.23 FSCTL_GET_REFS_VOLUME_DATA Request

Article08/24/2020

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

This message does not contain any parameters.

2.3.24 FSCTL_GET_REFS_VOLUME_DATA Reply

Article04/07/2025

The FSCTL_GET_REFS_VOLUME_DATA reply message returns the results of the FSCTL_GET_REFS_VOLUME_DATA request as an REFS_VOLUME_DATA_BUFFER element.

The REFS_VOLUME_DATA_BUFFER contains information on a volume.

Expand table

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
ByteCount																															
MajorVersion																															
MinorVersion																															
BytesPerPhysicalSector																															
VolumeSerialNumber																															
...																															
NumberSectors																															
...																															
TotalClusters																															
...																															
FreeClusters																															
...																															
TotalReserved																															
...																															
BytesPerSector																															
BytesPerCluster																															
MaximumSizeOfResidentFile																															
...																															
Reserved (80 bytes)																															
...																															
...																															

ByteCount (4 bytes): A 32-bit unsigned integer that contains the valid data length for this structure. **ByteCount** can be less than the size of this structure. Only the fields that entirely fit within the valid data

length for this structure, as defined by **ByteCount**, are valid.

MajorVersion (4 bytes): A 32-bit unsigned integer that contains the major version of the ReFS volume.

MinorVersion (4 bytes): A 32-bit unsigned integer that contains the minor version of the ReFS volume.

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

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 used to guarantee clusters are available at points when the file system can't properly report allocation failures.

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.

MaximumSizeOfResidentFile (8 bytes): A 64-bit unsigned integer that defines the maximum number of bytes a file can contain and be co-located with the file system metadata that describes the file (commonly known as resident files).

Reserved (80 bytes): 80 bytes which, if included, as per the **ByteCount** field, are reserved, have an undefined value, and are not interpreted.

This message also returns a status code as specified in section 2.2. Upon success, the status code returned directly by the function that processes this FSCTL is STATUS_SUCCESS. The most common error codes are listed in the following table.

Expand table

Error code	Meaning
STATUS_INVALID_PARAMETER 0xC000000D	The handle specified is not open.
STATUS_VOLUME_DISMOUNTED 0xC000026E	The specified volume is no longer mounted.

Error code	Meaning
STATUS_BUFFER_TOO_SMALL 0xC0000023	The output buffer is too small to contain a REFS_VOLUME_DATA_BUFFER structure.

2.3.25 FSCTL_GET_OBJECT_ID Request

Article08/24/2020

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

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

This message does not contain any additional data elements.

2.3.26 FSCTL_GET_OBJECT_ID Reply

Article04/07/2025

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

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

This message also returns a status code as specified in section 2.2. Upon success, the status code returned by the function that processes this FSCTL is STATUS_SUCCESS. The most common error codes are listed in the following table.

Expand table

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

2.3.27 FSCTL_GET_REPARSE_POINT Request

Article04/07/2025

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

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

This message does not contain any additional data elements.

2.3.28 FSCTL_GET_REPARSE_POINT Reply

Article04/07/2025

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

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

If the file system of the volume containing the specified file or directory does not support the use of reparse points, the request will not succeed. The error code returned in this situation MAY vary, depending on the file system.<29>

This message also returns a status code as specified in section 2.2. Upon success, the status code returned by the function that processes this FSCTL is STATUS_SUCCESS. The most common error codes are listed in the following table.

Expand table

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

2.3.29 FSCTL_GET_RETRIEVAL_POINTER_COUNT Request

Article08/24/2020

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

The STARTING_VCN_INPUT_BUFFER data element is as follows.

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
StartingVcn																															
...																															

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

2.3.30 FSCTL_GET_RETRIEVAL_POINTER_COUNT Reply

Article04/07/2025

The FSCTL_GET_RETRIEVAL_POINTER_COUNT reply message returns the results of the FSCTL_GET_RETRIEVAL_POINTER_COUNT request as a fixed size data element, RETRIEVAL_POINTER_COUNT, that specifies the number of extents on disk of a specific file.

The FSCTL_GET_RETRIEVAL_POINTER_COUNT reply returns the number of extents 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.<30>

The RETRIEVAL_POINTER_COUNT data element is as follows.

Expand table

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																															

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

This message also returns a status code as specified in section 2.2. Upon success, the status code returned by the function that processes this FSCTL is STATUS_SUCCESS. The most common error codes are listed in the following table.

Expand table

Error code	Meaning
STATUS_BUFFER_TOO_SMALL 0xC0000023	The output buffer is too small to contain a RETRIEVAL_POINTER_COUNT 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.

2.3.31 FSCTL_GET_RETRIEVAL_POINTERS Request

Article04/07/2025

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

The STARTING_VCN_INPUT_BUFFER data element is as follows.

Expand table

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
StartingVcn																															
...																															

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

2.3.32 FSCTL_GET_RETRIEVAL_POINTERS Reply

Article04/07/2025

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

The FSCTL_GET_RETRIEVAL_POINTERS reply returns the extent locations (that is, locations of allocated regions of disk space) of nonresident data. A file system MAY allow resident data, which is data that can be written to disk within the file's directory record. Because resident data requires no additional disk space allocation, no extent locations are associated with resident data. <31>

The RETRIEVAL_POINTERS_BUFFER data element is as follows.

Expand table

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

StartingVcn (8 bytes): A 64-bit signed integer that contains the starting virtual cluster number (VCN) returned by the FSCTL_GET_RETRIEVAL_POINTERS reply. This is not necessarily the VCN requested by the FSCTL_GET_RETRIEVAL_POINTERS request, as the file system driver might return the starting VCN of the extent containing the requested starting VCN. This value MUST be greater than or equal to 0.

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

2.3.32.1 EXTENTS

Article04/07/2025

The **EXTENTS** data element is as follows.

Expand table

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

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

This message also returns a status code as specified in section 2.2. Upon success, the status code returned by the function that processes this FSCTL is STATUS_SUCCESS. The most common error codes are listed in the following table.

Expand table

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

2.3.33 FSCTL_GET_RETRIEVAL_POINTERS_AND_REFCOUNT Request

Article12/14/2021

The FSCTL_GET_RETRIEVAL_POINTERS_AND_REFCOUNT request message requests that the server return a list of extents and their reference counts for the file or directory associated with the handle on which this **FSCTL** was invoked. The extents describe the mapping between **virtual cluster numbers (VCNs)** and **logical cluster numbers (LCNs)**. The reference count describes how many times these **logical cluster numbers (LCNs)** are being used within the **volume**. This request is most commonly used by deduplication utilities. This message contains a STARTING_VCN_INPUT_BUFFER data element.<32>

The STARTING_VCN_INPUT_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
StartingVcn																															
...																															

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

2.3.34 FSCTL_GET_RETRIEVAL_POINTERS_AND_REFCOUNT Reply

Article04/07/2025

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

The FSCTL_GET_RETRIEVAL_POINTERS_AND_REFCOUNT reply returns the extent locations (that is, locations of allocated regions of disk space) and their reference counts 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 or reference counts are associated with resident data.<33>

The RETRIEVAL_POINTERS_AND_REFCOUNT_BUFFER data element is as follows.

Expand table

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 EXTENT_AND_REFCOUNTS data elements in the Extents array. This number can be zero if there are no clusters allocated at (or beyond) the specified **StartingVcn**.

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

StartingVcn (8 bytes): A 64-bit signed integer that contains the starting **virtual cluster number (VCN)** returned by the FSCTL_GET_RETRIEVAL_POINTER_AND_REFCOUNT reply. This is not necessarily the VCN requested by the FSCTL_GET_RETRIEVAL_POINTERS request, as the file system driver might return the starting VCN of the extent containing the requested starting VCN. This value MUST be greater than or equal to 0.

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

2.3.34.1 EXTENT_AND_REFCOUNTS

Article02/10/2025

The EXTENT_AND_REFCOUNTS data element is as follows.

Expand table

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																															
...																															
ReferenceCount																															

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

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

ReferenceCount (4 bytes): A 32-bit unsigned integer that contains the reference count on the **logical cluster number (LCN)** at which the current extent begins on the **volume**. If no one else is using the corresponding LCN, the reference count will be 1.

This message also returns a status code as specified in section 2.2. Upon success, the status code returned by the function that processes this **FSCTL** is STATUS_SUCCESS. The most common error codes are listed in the following table.

Expand table

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

Error code	Meaning
STATUS_BUFFER_OVERFLOW 0x80000005	The output buffer filled before all the extents for this file were returned.

2.3.35 FSCTL_IS_PATHNAME_VALID Request

Article04/07/2025

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

The data element is as follows.

Expand table

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
PathNameLength																															
PathName (variable)																															
...																															

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.

2.3.36 FSCTL_IS_PATHNAME_VALID Reply

Article04/06/2021

This message returns the results of the FSCTL_IS_PATHNAME_VALID Request (section 2.3.35).

A STATUS_SUCCESS from this call means that the pathname is valid. An error means that the pathname is not valid.<34>

2.3.37 FSCTL_LMR_SET_LINK_TRACKING_INFORMATION Request

Article04/07/2025

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

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

- FSCTL_LMR_SET_LINK_TRACKING_INFORMATION Request for SMB
- FSCTL_LMR_SET_LINK_TRACKING_INFORMATION Request for SMB2

2.3.37.1 FSCTL_LMR_SET_LINK_TRACKING_INFORMATION Request for SMB

Article08/24/2020

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

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

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

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

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

2.3.37.2 FSCTL_LMR_SET_LINK_TRACKING_INFORMATION Request for SMB2

Article10/07/2024

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

Expand table

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

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

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

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

2.3.37.3 TARGET_LINK_TRACKING_INFORMATION_Buffer

Article08/24/2020

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.

2.3.37.3.1 TARGET_LINK_TRACKING_INFORMATION_Buffer_1

Article08/24/2020

If the **TargetLinkTrackingInformationLength** value is less than 36, the TARGET_LINK_TRACKING_INFORMATION_Buffer data element MUST be 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
NetBIOSName (variable)																															
...																															

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.

2.3.37.3.2 TARGET_LINK_TRACKING_INFORMATION_Buffer_2

Article04/07/2025

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

Expand table

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
Type																															
VolumeId (16 bytes)																															
...																															
...																															
ObjectId (16 bytes)																															
...																															
...																															
NetBIOSName (variable)																															
...																															

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.

Expand table

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

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

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

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

2.3.38 FSCTL_LMR_SET_LINK_TRACKING_INFORMATION Reply

Article04/07/2025

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 section 2.2. Upon success, the status code returned by the function that processes this FSCTL is STATUS_SUCCESS. The most common error codes are listed in the following table.

Expand table

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

2.3.39 FSCTL_MARK_HANDLE Request

Article04/27/2022

The FSCTL_MARK_HANDLE request is used to set specific operational state on the given file handle. This state is lost once the handle is closed.<35>

The MARK_HANDLE_INFO 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
CopyNumber																															
Unused																															
VolumeHandle																															
...																															
HandleInfo																															
Reserved																															

CopyNumber (4 bytes): A 32-bit unsigned integer that identifies, when reading from a file which resides on redundant media, which copy to read.

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

VolumeHandle (8 bytes): A 64-bit HANDLE that is not used and MUST be set to zero.

HandleInfo (4 bytes): A 32-bit unsigned integer containing flags to identify the request. Only one of the following values can be set:

Value	Meaning
MARK_HANDLE_READ_COPY 0x00000080	When a file resides on redundant media (ex: mirrored or RAID) this tells the file system that read operations on this handle should only come from the specified copy of data. When this state is not set a file system will return data from any copy available as it sees fit. This operation is typically used by scrubber applications that want to validate the contents of all copies of data for a given file.
MARK_HANDLE_NOT_READ_COPY 0x00000100	When a file resides on redundant media (ex: mirrored or RAID) this tells the file system that read operations on this handle may come from any copy of the data as the file system sees fit. This turns off reading from a specific copy.

Reserved (4 Bytes): A 32-bit field. This field is reserved. This field SHOULD be set to 0, and MUST be ignored.

2.3.40 FSCTL_MARK_HANDLE Reply

Article04/07/2025

This message returns the results of the FSCTL_MARK_HANDLE request.

The only data item this message returns is a status code, as specified in section 2.2. Upon success, the status code returned by the function that processes this FSCTL is STATUS_SUCCESS. The most common error codes are listed in the following table.

Expand table

Error code	Meaning
STATUS_INVALID_PARAMETER 0xC000000D	This status is returned if: • HandleInfo contains any flag other than one and only one of either MARK_HANDLE_READ_COPY or MARK_HANDLE_NOT_READ_COPY • The file was opened for cached IO • The specified copy number is greater than the number of available redundant copies
STATUS_DIRECTORY_NOT_SUPPORTED 0xC000047C	This operation is not supported on directory files.
STATUS_NOT_REDUNDANT_STORAGE 0xC0000479	This operation is only supported on redundant media.
STATUS_COMPRESSED_FILE_NOT_SUPPORTED 0xC000047B	This operation is not supported on compressed files.

2.3.41 FSCTL_OFFLOAD_READ Request

Article02/10/2025

The FSCTL_OFFLOAD_READ Request message requests that the server perform an Offload Read operation to a specified portion of a file on a target volume. On the client side, this request is received, processed, and sent down to an intelligent storage subsystem that generates and returns a Token in an FSCTL_OFFLOAD_READ Reply (section 2.3.42) message. This Token logically represents the data to be read and can be used with an FSCTL_OFFLOAD_WRITE Request (section 2.3.43) and an FSCTL_OFFLOAD_WRITE Reply (section 2.3.44) pair to complete the data movement.<36>

The request message contains an **FSCTL_OFFLOAD_READ_INPUT** data element, as follows.

Expand table

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
Size																															
Flags																															
TokenTimeToLive																															
Reserved																															
FileOffset																															
...																															
CopyLength																															
...																															

Size (4 bytes): A 32-bit unsigned integer that indicates the size, in bytes, of this data element.

Flags (4 bytes): A 32-bit unsigned integer that indicates the flags to be set for this operation. Currently, no flags are defined. This field SHOULD be set to 0x00000000 and MUST be ignored.

TokenTimeToLive (4 bytes): A 32-bit unsigned integer that contains the requested Time to Live (TTL) value in milliseconds for the generated Token. This value MUST be greater than or equal to 0x00000000. A value of 0x00000000 represents a default TTL interval.<37>

Reserved (4 bytes): A 32-bit unsigned integer field that is reserved. This field SHOULD be set to 0x00000000 and MUST be ignored.

FileOffset (8 bytes): A 64-bit unsigned integer that contains the file offset, in bytes, of the start of a range of bytes in a file from which to generate the Token. The value of this field MUST be greater than or equal to 0x0000000000000000 and MUST be aligned to a logical sector boundary on the volume.

CopyLength (8 bytes): A 64-bit unsigned integer that contains the size, in bytes, of the requested range of the file from which to generate the Token. The value of this field MUST be greater than or equal to 0x0000000000000000 and MUST be aligned to a logical sector boundary on the volume.<38>

2.3.42 FSCTL_OFFLOAD_READ Reply

Article04/07/2025

The FSCTL_OFFLOAD_READ Reply message returns the results of the FSCTL_OFFLOAD_READ Request (section 2.3.41).

The **FSCTL_OFFLOAD_READ_OUTPUT** data element is as follows.

Expand table

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
Size																															
Flags																															
TransferLength																															
...																															
Token (512 bytes)																															
...																															
...																															

Size (4 bytes): A 32-bit unsigned integer that indicates the size, in bytes, of the returned data element.

Flags (4 bytes): A 32-bit unsigned integer that indicates which flags were returned for this operation. Possible values for the flags follow. All unused bits are reserved for future use, SHOULD be set to 0, and MUST be ignored.

Expand table

Value	Meaning
OFFLOAD_READ_FLAG_ALL_ZERO_BEYOND_CURRENT_RANGE 0x00000001	The data beyond the current range is logically equivalent to zero.

TransferLength (8 bytes): A 64-bit unsigned integer that contains the amount, in bytes, of data that the Token logically represents. This value indicates a contiguous region of the file from the beginning of the requested offset in the **FileOffset** field in the FSCTL_OFFLOAD_READ_INPUT data element (section 2.3.41). This value can be smaller than the **CopyLength** field specified in the FSCTL_OFFLOAD_READ_INPUT data element, which indicates that less data was logically represented (logically read) with the Token than was requested. The value of this field MUST be greater than 0x0000000000000000 and MUST be aligned to a logical sector boundary on the volume.

Token (512 bytes): A STORAGE_OFFLOAD_TOKEN (section 2.1.11) structure that contains the generated Token to be used as a representation of the data contained within the portion of the file specified in the

FSCTL_OFFLOAD_READ_INPUT data element at the time of the FSCTL_OFFLOAD_READ operation. The contents of this field MUST NOT be modified during subsequent operations.<39>

This message returns a status code as specified in section 2.2. Upon success, the status code returned by the function that processes this FSCTL is STATUS_SUCCESS. The most common error codes are listed in the following table.

Expand table

Error code	Meaning
STATUS_INVALID_DEVICE_REQUEST 0xC0000010	The file system does not support offload operations.
STATUS_INVALID_PARAMETER 0xC000000D	At least one of the following assertions is true: • The target file is smaller than the logical sector size. • The **FileOffset** field is not a multiple of the logical sector size of the volume. • The **CopyLength** field is not a multiple of the logical sector size of the volume. • The **Size** field is not equivalent to the size of an FSCTL_OFFLOAD_READ_INPUT data element. • Adding the **FileOffset** and **CopyLength** fields results in the overflow of a 64-bit value.
STATUS_OFFLOAD_READ_FILE_NOT_SUPPORTED 0xC000A2A3	Offload operations cannot be performed on: • Compressed Files • Sparse Files • Encrypted Files • File System Metadata Files
STATUS_NOT_SUPPORTED 0xC00000BB	The file system indicates that the volume does not support the Offload Read operation.
STATUS_OFFLOAD_READ_FLT_NOT_SUPPORTED 0xC000A2A1	A file system filter on the server has not opted in for Offload Read support.
STATUS_FILE_DELETED 0xC0000123	The specified data stream is not valid.
STATUS_FILE_CLOSED 0xC0000128	The specified file handle is closed.
STATUS_END_OF_FILE 0xC0000011	The file read starts beyond the End Of the File (EOF).<40>
STATUS_INSUFFICIENT_RESOURCES	There were insufficient resources to complete the operation.

Error code	Meaning
0xC000009A	
STATUS_BUFFER_TOO_SMALL 0xC0000023	The input buffer is too small to contain an FSCTL_OFFLOAD_READ_INPUT data element. or The output buffer is too small to contain an FSCTL_OFFLOAD_READ_OUTPUT data element.
STATUS_DEVICE_FEATURE_NOT_SUPPORTED 0xC0000463	The storage device does not support offload read.

2.3.43 FSCTL_OFFLOAD_WRITE Request

The FSCTL_OFFLOAD_WRITE Request message requests that the server perform an Offload Write operation to a specified portion of a file on a target volume, providing a Token to the server that indicates what data is to be logically written. On the server side, this request is received, processed, and sent to an intelligent storage subsystem that processes the Token and determines whether it can perform the data movement to the requested portion of the file. The Token is generated by an intelligent storage subsystem through an FSCTL_OFFLOAD_READ Request (section 2.3.41) or is constructed as a well-known Token type such as STORAGE_OFFLOAD_TOKEN in section 2.1.11.<41><42>

The request message contains an **FSCTL_OFFLOAD_WRITE_INPUT** data element, as follows:

Expand table

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
Size																															
Flags																															
FileOffset																															
...																															
CopyLength																															
...																															
TransferOffset																															
...																															
Token (512 bytes)																															
...																															
...																															

Size (4 bytes): A 32-bit unsigned integer that indicates the size, in bytes, of this data element.

Flags (4 bytes): A 32-bit unsigned integer that indicates the flags to be set for this operation. Currently, no flags are defined. This field SHOULD be set to 0x00000000 and MUST be ignored.

FileOffset (8 bytes): A 64-bit unsigned integer that contains the file offset, in bytes, of the start of a range of bytes in a file at which to begin writing the data logically represented by the Token. The value of this field MUST be greater than or equal to 0x0000000000000000 and MUST be aligned to a logical sector boundary on the volume.

CopyLength (8 bytes): A 64-bit unsigned integer that contains the size, in bytes, of the requested range of the file to write the data logically represented by the Token. The value of this field MUST be greater than or equal to 0x0000000000000000 and MUST be aligned to a logical sector boundary on the volume. This value can be smaller than the size of the data logically represented by the Token.

TransferOffset (8 bytes): A 64-bit unsigned integer that contains the file offset, in bytes, relative to the front of a region of data logically represented by the Token at which to start writing. The value of this field MUST be greater than or equal to 0x0000000000000000 and MUST be aligned to a logical sector boundary on the volume.

Token (512 bytes): A STORAGE_OFFLOAD_TOKEN (section 2.1.11) structure that contains the generated (or constructed) Token to be used as a representation of the data to be logically written. The contents of this field MUST NOT be modified during subsequent operations.

Last updated on 11/21/2025

2.3.44 FSCTL_OFFLOAD_WRITE Reply

Article04/23/2024

The FSCTL_OFFLOAD_WRITE Reply message returns the results of the FSCTL_OFFLOAD_WRITE Request (section 2.3.43).

The **FSCTL_OFFLOAD_WRITE_OUTPUT** data element is as follows.

Expand table

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
Size																															
Flags																															
LengthWritten																															
...																															

Size (4 bytes): A 32-bit unsigned integer that indicates the size, in bytes, of the returned data element.

Flags (4 bytes): A 32-bit unsigned integer that indicates which flags were returned for this operation. Currently, no flags are defined. This field SHOULD be set to 0x00000000 and MUST be ignored.

LengthWritten (8 bytes): A 64-bit unsigned integer that contains the amount, in bytes, of data that was written. The value of this field MUST be greater than or equal to zero and MUST be aligned to a logical sector boundary on the volume. This value can be smaller than the **CopyLength** field specified in the FSCTL_OFFLOAD_WRITE_INPUT data element. A smaller value indicates that less data was logically written with the specified Token than was requested. This field MUST NOT be greater than the **CopyLength** field specified in the FSCTL_OFFLOAD_WRITE_INPUT data element, meaning it is incorrect to copy more than what was requested<43>.

This message returns a status code as specified in section 2.2. Upon success, the status code returned by the function that processes this FSCTL is STATUS_SUCCESS. The most common error codes are listed in the following table.

Expand table

Error code	Meaning
STATUS_INVALID_DEVICE_REQUEST 0xC0000010	The file system does not support offload operations.
STATUS_INVALID_PARAMETER 0xC000000D	At least one of the following assertions is true: • The target file is smaller than the logical sector size. • The **FileOffset** field is not a multiple of the logical sector size of the volume. • The **CopyLength** field is not a multiple of the logical sector size of the volume.

Error code	Meaning
	• The **TransferOffset** field is not a multiple of the logical sector size of the volume. • The **FileOffset** field is greater than the Valid Data Length (VDL) for the file. • The **Size** field is not equivalent to the size of an FSCTL_OFFLOAD_WRITE_INPUT data element. • Adding the **FileOffset** and **CopyLength** fields results in the overflow of a 64-bit value.
STATUS_OFFLOAD_WRITE_FILE_NOT_SUPPORTED 0xC000A2A4	Offload operations cannot be performed on: • Compressed Files • Sparse Files • Encrypted Files • File System Metadata Files
STATUS_NOT_SUPPORTED 0xC00000BB	The file system indicates that the volume does not support the Offload Write operation.
STATUS_OFFLOAD_WRITE_FLT_NOT_SUPPORTED 0xC000A2A2	A file system filter on the server has not opted in for Offload Write support.
STATUS_FILE_DELETED 0xC0000123	The specified data stream was not valid.
STATUS_FILE_CLOSED 0xC0000128	The specified file handle is closed.
STATUS_END_OF_FILE 0xC0000011	The file offset for the write is beyond the End Of the File (EOF).
STATUS_MEDIA_WRITE_PROTECTED 0xC00000A2	The volume is read only.
STATUS_INSUFFICIENT_RESOURCES 0xC000009A	There were insufficient resources to complete the operation.
STATUS_BUFFER_TOO_SMALL 0xC0000023	The input buffer is too small to contain an FSCTL_OFFLOAD_WRITE_INPUT data element. or The output buffer is too small to contain an FSCTL_OFFLOAD_WRITE_OUTPUT data element.
STATUS_DEVICE_FEATURE_NOT_SUPPORTED 0xC0000463	The storage device does not support Offload Write.
STATUS_DEVICE_UNREACHABLE 0xC0000464	Data cannot be moved by Offload Write because the source device cannot communicate with the destination device.

Error code	Meaning
STATUS_INVALID_TOKEN 0xC0000465L	The token representing the data is invalid or expired.

2.3.45 FSCTL_PIPE_PEEK Request

Article08/24/2020

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

2.3.46 FSCTL_PIPE_PEEK Reply

Article05/28/2021

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

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
NamedPipeState																															
ReadDataAvailable																															
NumberOfMessages																															
MessageLength																															
Data (variable)																															
...																															

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

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

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

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

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

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

This message returns a status code as specified in section 2.2. Upon success, the status code returned by the function that processes this FSCTL is STATUS_SUCCESS. The most common error codes are listed in the following table.

Error code	Meaning
STATUS_PIPE_DISCONNECTED 0xC00000B0	The specified named pipe is in the disconnected state.

Error code	Meaning
STATUS_INVALID_PIPE_STATE 0xC00000AD	The data cannot be read in the current state of the specified pipe.
STATUS_PIPE_BROKEN 0xC000014B	The pipe operation has failed because the other end of the pipe has been closed.
STATUS_INVALID_USER_BUFFER 0xC00000E8	An exception was raised while accessing a user buffer.
STATUS_INSUFFICIENT_RESOURCES 0xC000009A	There were insufficient resources to complete the operation.
STATUS_INVALID_DEVICE_REQUEST 0xC0000010	The type of the handle is not a pipe.
STATUS_BUFFER_OVERFLOW 0x80000005	The data was too large for the specified buffer. This is a warning, not an error. Response contains information including available data length and data that fits into the buffer.

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

2.3.47 FSCTL_PIPE_TRANSCEIVE Request

06/10/2025

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

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

2.3.48 FSCTL_PIPE_TRANSCEIVE Reply

Article04/07/2025

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

This message returns a status code as specified in section 2.2. Upon success, the status code returned by the function that processes this FSCTL is STATUS_SUCCESS. The most common error codes are listed in the following table.

Expand table

Error code	Meaning
STATUS_PIPE_DISCONNECTED 0xC00000B0	The specified named pipe is in the disconnected state.
STATUS_INVALID_PIPE_STATE 0xC00000AD	The named pipe is not in the connected state or not in the full-duplex message mode.
STATUS_PIPE_BUSY 0xC00000AE	The named pipe contains unread data.
STATUS_INVALID_USER_BUFFER 0xC00000E8	An exception was raised while accessing a user buffer.
STATUS_INSUFFICIENT_RESOURCES 0xC000009A	There were insufficient resources to complete the operation.
STATUS_INVALID_DEVICE_REQUEST 0xC0000010	The type of the handle is not a pipe.
STATUS_BUFFER_OVERFLOW 0x80000005	The data was too large to fit into the specified buffer.

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

2.3.49 FSCTL_PIPE_WAIT Request

Article10/30/2020

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.

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
Timeout																															
...																															
NameLength																															
TimeoutSpecified								Padding								Name (variable)															
...																															

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

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

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

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

Padding (1 byte): The client SHOULD set this field to 0x00, and the server MUST ignore it.

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

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

2.3.50 FSCTL_PIPE_WAIT Reply

Article04/07/2025

This message returns the results of the FSCTL_PIPE_WAIT request.

The only data item this message returns is a status code, as specified in section 2.2. Upon success, the status code returned by the function that processes this FSCTL is STATUS_SUCCESS. The most common error codes are listed in the following table.

Expand table

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

2.3.51 FSCTL_QUERY_ALLOCATED_RANGES Request

Article04/07/2025

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

The FILE_ALLOCATED_RANGE_BUFFER data element is as follows.

Expand table

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
FileOffset																															
...																															
Length																															
...																															

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

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

2.3.52 FSCTL_QUERY_ALLOCATED_RANGES Reply

Article04/07/2025

The FSCTL_QUERY_ALLOCATED_RANGES Reply message returns the results of the FSCTL_QUERY_ALLOCATED_RANGES Request (section 2.3.51).

This message MUST return an array of zero or more FILE_ALLOCATED_RANGE_BUFFER data elements. The number of FILE_ALLOCATED_RANGE_BUFFER elements returned is computed by dividing the size of the returned output buffer (from either SMB or SMB2, the lower-layer protocol that carries the FSCTL) by the size of the FILE_ALLOCATED_RANGE_BUFFER element. Ranges returned MUST intersect the range specified in the FSCTL_QUERY_ALLOCATED_RANGES Request. Zero FILE_ALLOCATED_RANGE_BUFFER data elements MUST be returned when the file has no allocated ranges.<44>

The FILE_ALLOCATED_RANGE_BUFFER data element is as follows.

Expand table

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
FileOffset																															
...																															
Length																															
...																															

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.<45> 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.<46>

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

This message returns a status code as specified in section 2.2. Upon success, the status code returned by the function that processes this FSCTL is STATUS_SUCCESS. The most common error codes are listed in the following table.

Expand table

Error code	Meaning
STATUS_INVALID_PARAMETER 0xC000000D	The handle is not to a file, or the size of the input buffer is less than the size of a FILE_ALLOCATED_RANGE_BUFFER structure, or the given **FileOffset** field value is less

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

2.3.53 FSCTL_QUERY_FAT_BPB Request

Article04/07/2025

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

This message does not contain any additional data elements.

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

2.3.54 FSCTL_QUERY_FAT_BPB Reply

06/10/2025

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

This message also returns a status code as specified in section 2.2. Upon success, the status code returned by the function that processes this FSCTL is STATUS_SUCCESS. The most common error codes are listed in the following table.

Expand table

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

2.3.55 FSCTL_QUERY_FILE_REGIONS Request

Article12/14/2021

The FSCTL_QUERY_FILE_REGIONS request message requests that the server return a list of file regions, based on a specified usage parameter, for the file associated with the handle on which this FSCTL was invoked. This message contains an optional FILE_REGION_INPUT data element. If no FILE_REGION_INPUT parameter is specified, information for the entire size of the file is returned.

A FILE_REGION_INPUT 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
FileOffset																															
...																															
Length																															
...																															
DesiredUsage																															
Reserved																															

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.

Length (8 bytes): A 64-bit signed integer that contains the size, in bytes, of the range.

DesiredUsage (4 bytes): A 32-bit unsigned integer that indicates usage parameters for this operation. The following table provides the currently defined usage parameters.

Value	Meaning
FILE_REGION_USAGE_VALID_CACHED_DATA 0x00000001	Information about the valid data length for the specified file and file range in the cache will be returned.<47>
FILE_REGION_USAGE_VALID_NONCACHED_DATA 0x00000002	Information about the valid data length for the specified file and file range on disk will be returned.<48>
All other values	If a FILE_REGION_INPUT object is specified in FSCTL_QUERY_FILE_REGION, then any other value will return STATUS_INVALID_PARAMETER.

Reserved (4 bytes): A 32-bit unsigned integer that is reserved. This field SHOULD be 0x00000000 and MUST be ignored.

2.3.56 FSCTL_QUERY_FILE_REGIONS Reply

Article04/07/2025

The FSCTL_QUERY_FILE_REGIONS reply message returns the results of the FSCTL_QUERY_FILE_REGIONS Request as a variably sized data element, FILE_REGION_OUTPUT, which contains one or more FILE_REGION_INFO elements that contain the ranges computed as a result of the desired usage.

A FILE_REGION_OUTPUT data element is as follows.

Expand table

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
Flags																															
TotalRegionEntryCount																															
RegionEntryCount																															
Reserved																															
Region (variable)																															
...																															
...																															

Flags (4 bytes): A 32-bit unsigned integer that indicates the flags for this operation. No flags are currently defined, thus this field SHOULD be set to 0x00000000 and MUST be ignored.

TotalRegionEntryCount (4 bytes): A 32-bit unsigned integer that indicates the total number of regions that could be returned.

RegionEntryCount (4 bytes): A 32-bit unsigned integer that indicates the number of regions that were actually returned and which are contained in this structure.

Reserved (4 bytes): A 32-bit unsigned integer that is reserved. This field SHOULD be set to 0x00000000 and MUST be ignored.

Region (variable): One or more FILE_REGION_INFO structures, as specified in section 2.3.56.1, that contain information on the desired ranges based on the desired usage indicated by the **DesiredUsage** field.

This message returns a status code as specified in section 2.2. Upon success, the status code returned by the function that processes this FSCTL is STATUS_SUCCESS. The most common error codes are listed in the following table.

Expand table

Error code	Meaning
STATUS_BUFFER_TOO_SMALL	The input buffer is too small to contain a FILE_REGION_INPUT structure, or the output buffer is too small to contain a FILE_REGION_OUTPUT structure.

Error code	Meaning
0xC0000023	
STATUS_BUFFER_OVERFLOW 0x80000005	The output buffer was filled before all the desired regions for this file were returned.
STATUS_INVALID_PARAMETER 0xC000000D	A specified file region is invalid, or the specified desired usage flag is invalid, or the given handle is not for a file (but for a directory or volume instead).

2.3.56.1 FILE_REGION_INFO

Article09/20/2023

The **FILE_REGION_INFO** structure contains a computed region of a file based on a desired usage. This structure is used to store region information for the FSCTL_QUERY_FILE_REGIONS reply message, with the FILE_REGION_OUTPUT structure containing one or more FILE_REGION_INFO structures.

A FILE_REGION_INFO 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
FileOffset																															
...																															
Length																															
...																															
DesiredUsage																															
Reserved																															

FileOffset (8 bytes): A 64-bit signed integer that contains the file offset, in bytes, of the region.

Length (8 bytes): A 64-bit signed integer that contains the size, in bytes, of the region.

DesiredUsage (4 bytes): A 32-bit unsigned integer that indicates the usage for the given region of the file.

Value	Meaning
0x00000000	The given range is invalid. It does not match the criteria of the requested **DesiredUsage** as specified in section 2.3.55.
FILE _USAGE_VALID_CACHED_DATA 0x00000001	Defines those regions of the file that exists before VDL as it exists in the cache manager.<49>
FILE _USAGE_VALID_NONCACHED_DATA 0x00000002	Defines those regions of the files that exist before VDL on the storage device.<50>

Reserved (4 bytes): A 32-bit unsigned integer field that is reserved. This field SHOULD be set to 0x00000000 and MUST be ignored.

2.3.57 FSCTL_QUERY_ON_DISK_VOLUME_INFO Request

Article04/07/2025

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

This message does not contain any additional data elements.

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

2.3.58 FSCTL_QUERY_ON_DISK_VOLUME_INFO Reply

Article04/27/2022

This message returns the results of the FSCTL_QUERY_ON_DISK_VOLUME_INFO request (section 2.3.57) as a FSCTL_QUERY_ON_DISK_VOLUME_INFO_BUFFER structure.

0	1	2	3	4	5	6	7	8	9	10	1	2	3	4	5	6	7	8	9	20	1	2	3	4	5	6	7	8	9	30	1
DirectoryCount																															
...																															
FileCount																															
...																															
FsFormatMajVersion																FsFormatMinVersion															
FsFormatName (24 bytes)																															
...																															
...																															
FormatTime																															
...																															
LastUpdateTime																															
...																															
CopyrightInfo (68 bytes)																															
...																															
...																															
AbstractInfo (68 bytes)																															
...																															
...																															
FormattingImplementationInfo (68 bytes)																															
...																															
...																															
LastModifyingImplementationInfo (68 bytes)																															
...																															
...																															

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

For UDF file systems with a virtual allocation table, this information is available only if the UDF revision of the volume is greater than 1.50.<51>

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

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

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

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

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

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

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

CopyrightInfo (68 bytes): A Unicode string containing any copyright notifications associated with the volume. This information is implementation-specific and will be padded with NULLs.<52>

AbstractInfo (68 bytes): A Unicode string containing any abstract information written on the volume. This information is implementation-specific and will be padded with NULLs.<53>

FormattingImplementationInfo (68 bytes): A Unicode string containing the operating system version that the volume was formatted by. This information is implementation-specific and will be padded with NULLs.<54>

LastModifyingImplementationInfo (68 bytes): A Unicode string containing the operating system version that the volume was last modified by. This information is implementation-specific and will be padded with NULLs.<55>

This message returns a status code as specified in section 2.2. Upon success, the status code returned by the function that processes this FSCTL is STATUS_SUCCESS. The most common error codes are listed in the following table.

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

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

2.3.59 FSCTL_QUERY_SPARING_INFO Request

Article04/07/2025

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

This message does not contain any additional data elements.

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

2.3.60 FSCTL_QUERY_SPARING_INFO Reply

Article04/07/2025

This message returns the results of the FSCTL_QUERY_SPARING_INFO request (section 2.3.59) as a FSCTL_QUERY_SPARING_BUFFER structure.

Expand table

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
SparingUnitBytes																															
SoftwareSparing								Reserved																							
TotalSpareBlocks																															
FreeSpareBlocks																															

SparingUnitBytes (4 bytes): A 32-bit unsigned integer that contains the size, in bytes, of a sparing packet, which is the same as the underlying error check and correction (ECC) block size of the media. For more information, see [UDF].

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

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

TotalSpareBlocks (4 bytes): A 32-bit unsigned integer that contains the total number of blocks allocated for sparing.

FreeSpareBlocks (4 bytes): A 32-bit unsigned integer that contains the number of blocks available for sparing.

This message returns a status code as specified in section 2.2. Upon success, the status code returned by the function that processes this FSCTL is STATUS_SUCCESS. The most common error codes are listed in the following table.

Expand table

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

2.3.61 FSCTL_READ_FILE_USN_DATA Request

Article04/07/2025

This message requests that the server return the most recent change journal USN for the file or directory associated with the handle on which this FSCTL was invoked. This message contains an optional READ_FILE_USN_DATA data element.<56>

The READ_FILE_USN_DATA data element is as follows.

Expand table

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
MinMajorVersion																MaxMajorVersion															

MinMajorVersion (2 bytes): A 16-bit unsigned integer that contains the minimum major version of records returned in the results of this request.<57>

MaxMajorVersion (2 bytes): A 16-bit unsigned integer that contains the maximum major version of records returned in the results of this request.<58>

2.3.62 FSCTL_READ_FILE_USN_DATA Reply

Article04/07/2025

The FSCTL_READ_FILE_USN_DATA reply message returns the results of the FSCTL_READ_FILE_USN_DATA request as a USN_RECORD_V2 or a USN_RECORD_V3. Both forms of reply message begin with a USN_RECORD_COMMON_HEADER, which can be used to determine the form of the full reply message.

This message returns a status code as specified in section 2.2. Upon success, the status code returned by the function that processes this FSCTL is STATUS_SUCCESS. The most common error codes are listed in the following table.

Expand table

Error code	Meaning
STATUS_INVALID_PARAMETER 0xC000000D	The handle is not to a file, directory or if invalid **MinMajorVersion** and **MaxMajorVersion** values are specified. .
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.

2.3.62.1 USN_RECORD_COMMON_HEADER

Article04/27/2022

The USN_RECORD_COMMON_HEADER 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
RecordLength																															
MajorVersion																MinorVersion															

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.<59>

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).<60>

2.3.62.2 USN_RECORD_V2

Article04/27/2022

The USN_RECORD_V2 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
RecordLength																															
MajorVersion																MinorVersion															
FileReferenceNumber																															
...																															
ParentFileReferenceNumber																															
...																															
Usn																															
...																															
TimeStamp																															
...																															
Reason																															
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 a USN_RECORD_V2, the major version number is 2.

MinorVersion (2 bytes): A 16-bit unsigned integer that contains the minor version of the change journal software for this record. For a USN_RECORD_V2, the minor version number is 0 (zero).

FileReferenceNumber (8 bytes): The 64-bit file ID, as specified in section 2.1.9, of the file or directory for which this record notes changes.

ParentFileReferenceNumber (8 bytes): The 64-bit file ID, as specified in section 2.1.9, of the directory on which the file or directory that is associated with this record is located.

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

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

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

Value	Meaning
USN_REASON_BASIC_INFO_CHANGE 0x00008000	A user has either changed one or more files or directory attributes (such as read-only, hidden, archive, or sparse) or one or more time stamps.
USN_REASON_CLOSE 0x80000000	The file or directory is closed.
USN_REASON_COMPRESSION_CHANGE 0x00020000	The compression state of the file or directory is changed from (or to) compressed.
USN_REASON_DATA_EXTEND 0x00000002	The file or directory is extended (added to).
USN_REASON_DATA_OVERWRITE 0x00000001	The data in the file or directory is overwritten.
USN_REASON_DATA_TRUNCATION 0x00000004	The file or directory is truncated.
USN_REASON_EA_CHANGE 0x00000400	The user made a change to the extended attributes of a file or directory. These NTFS file system attributes are not accessible to nonnative applications. This USN reason does not appear under normal system usage, but can appear if an application or utility bypasses the Win32 API and uses the native API to create or modify extended attributes of a file or directory.
USN_REASON_ENCRYPTION_CHANGE 0x00040000	The file or directory is encrypted or decrypted.
USN_REASON_FILE_CREATE 0x00000100	The file or directory is created for the first time.

Value	Meaning
USN_REASON_FILE_DELETE 0x00000200	The file or directory is deleted.
USN_REASON_HARD_LINK_CHANGE 0x00010000	A hard link is added to (or removed from) the file or directory.
USN_REASON_INDEXABLE_CHANGE 0x00004000	A user changes the FILE_ATTRIBUTE_NOT_CONTEXT_INDEXED attribute. That is, the user changes the file or directory from one in which content can be indexed to one in which content cannot be indexed, or vice versa.
USN_REASON_NAMED_DATA_EXTEND 0x00000020	The one (or more) named data stream for a file is extended (added to).
USN_REASON_NAMED_DATA_OVERWRITE 0x00000010	The data in one (or more) named data stream for a file is overwritten.
USN_REASON_NAMED_DATA_TRUNCATION 0x00000040	One (or more) named data stream for a file is truncated.
USN_REASON_OBJECT_ID_CHANGE 0x00080000	The object identifier of a file or directory is changed.
USN_REASON_RENAME_NEW_NAME 0x00002000	A file or directory is renamed, and the file name in the USN_RECORD structure is the new name.
USN_REASON_RENAME_OLD_NAME 0x00001000	The file or directory is renamed, and the file name in the USN_RECORD structure is the previous name.
USN_REASON_REPARSE_POINT_CHANGE 0x00100000	The reparse point that is contained in a file or directory is changed, or a reparse point is added to (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.
USN_REASON_INTEGRITY_CHANGE 0x00800000	A change is made in the integrity status of a file or directory.

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

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

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

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

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

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

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

The fields **Reason**, **TimeStamp**, **SourceInfo**, and **SecurityId** for a USN RECORD element returned by this FSCTL MUST all be set to 0.<61>

2.3.62.3 USN_RECORD_V3

Article02/10/2025

The **USN_RECORD_V3** element is as follows.

Expand table

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
RecordLength																															
MajorVersion																MinorVersion															
FileReferenceNumber (16 bytes)																															
...																															
...																															
ParentFileReferenceNumber (16 bytes)																															
...																															
...																															
Usn																															
...																															
TimeStamp																															
...																															
Reason																															
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 a USN_RECORD_V3, the major version number is 3.

MinorVersion (2 bytes): A 16-bit unsigned integer that contains the minor version of the change journal software for this record. For a USN_RECORD_V3, the minor version number is 0 (zero).

FileReferenceNumber (16 bytes): The 128-bit file ID, as specified in section 2.1.10, of the file or directory for which this record notes changes.

ParentFileReferenceNumber (16 bytes): The 128-bit file ID, as specified in section 2.1.10, of the directory on which the file or directory that is associated with this record is located.

The fields **Usn**, **TimeStamp**, **Reason**, **SourceInfo**, **SecurityId**, **FileAttributes**, **FileNameLength**, **FileNameOffset**, and **FileName** for a USN RECORD_V3 element are as described for a USN_RECORD_V2 element; see section 2.3.62.2.

2.3.63 FSCTL_RECALL_FILE Request

Article04/07/2025

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

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

This message does not contain any additional data elements.

2.3.64 FSCTL_RECALL_FILE Reply

Article04/07/2025

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 section 2.2. Upon success, the status code returned by the function that processes this FSCTL is STATUS_SUCCESS. The most common error codes are listed in the following table.

Expand table

Error code	Meaning
STATUS_ACCESS_DENIED 0xC0000022	The file is set to not allow recall.
ERROR_INVALID_FUNCTION 0x00000001	The Remote Storage option is not installed.
STATUS_NOT_SUPPORTED 0xC00000BB	The request is not supported.
STATUS_INVALID_DEVICE_REQUEST 0xC0000010	The supplied handle is not that of a file.

2.3.65 FSCTL_REFS_STREAM_SNAPSHOT_MANAGEMENT Request

Article04/27/2022

The FSCTL_REFS_STREAM_SNAPSHOT_MANAGEMENT request message requests that the server perform a specific stream snapshot operation on a given data stream contained in a file. The operation performed is dependent on the value defined in REFS_STREAM_SNAPSHOT_OPERATION. The request message takes the form of a REFS_STREAM_SNAPSHOT_MANAGEMENT_INPUT_BUFFER structure.

The REFS_STREAM_SNAPSHOT_MANAGEMENT_INPUT_BUFFER 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
Operation																															
SnapshotNameLength																OperationInputBufferLength															
Reserved																															
...																															
...																															
...																															
NameAndInputBuffer (variable)																															
...																															

Operation (4 bytes): This field specifies the operation and MUST contain one of the following values:

Value	Meaning
REFS_STREAM_SNAPSHOT_OPERATION_INVALID 0x00000000	All requests with this operational code MUST be failed by the server.
REFS_STREAM_SNAPSHOT_OPERATION_CREATE 0x00000001	This request message requests the server create a new snapshot of the UNICODE name contained within **NameAndInputBuffer**, saving a point-in-time view of the data stream represented by the handle the request is being sent on.
REFS_STREAM_SNAPSHOT_OPERATION_LIST 0x00000002	This request message requests the server return a list of all snapshots of the set containing the data stream represented by the handle the request is being sent on, and matching a given regular expression query string contained in NameAndInputBuffer.

Value	Meaning
REFS_STREAM_SNAPSHOT_OPERATION_QUERY_DELTAS 0x00000003	This request message requests the server return a list of all metadata extents that have incurred modifying operations between the data stream represented by the handle the request is being sent on, and the data stream represented by the UNICODE name contained in NameAndInputBuffer. The data stream represented by the handle must be of a newer creation time than the data stream represented by the UNICODE name.
REFS_STREAM_SNAPSHOT_OPERATION_REVERT 0x00000004	This request message requests the server revert the data stream represented by the handle the request is being sent on to a point-in-time snapshot view represented by the UNICODE name contained within NameAndInputBuffer.
REFS_STREAM_SNAPSHOT_OPERATION_SET_SHADOW_BTREE 0x00000005	This request message requests the server create a shadow data stream on the data stream represented by the handle the request is being sent on.
REFS_STREAM_SNAPSHOT_OPERATION_CLEAR_SHADOW_BTREE 0x00000006	This request message requests the server remove a shadow data stream on the data stream represented by the handle the request is being sent on.
REFS_STREAM_SNAPSHOT_OPERATION_MAX 0x00000006	The maximum operational code supported by the server. All operational codes larger than this numerical value will be failed.

SnapshotNameLength (2 bytes): An unsigned integer representing the length in bytes of the unicode name contained within NameAndInputBuffer field. If no such name is present in the message, then this value is set to zero.

OperationInputBufferLength (2 bytes): An unsigned integer representing the length in bytes of the operational control structure present in the message and contained within **NameAndInputBuffer** field. If no such control structure is present in the message, then this value is set to zero.

Reserved (16 bytes): This field MUST be set to zero and MUST be ignored.

NameAndInputBuffer (variable): An array of bytes optionally containing a unicode name as well as an operational control buffer. When a unicode name is present, it is located immediately within the first byte of **NameAndInputBuffer**. When an operational control buffer is present, it is located at the next quad aligned boundary past the end of the unicode name. If no such unicode name is present, then the operational control buffer is located at the first byte of **NameAndInputBuffer**.

The following **Operation** codes require a unicode name to be present:

- REFS_STREAM_SNAPSHOT_OPERATION_CREATE
- REFS_STREAM_SNAPSHOT_OPERATION_LIST
- REFS_STREAM_SNAPSHOT_OPERATION_QUERY_DELTAS

- REFS_STREAM_SNAPSHOT_OPERATION_REVERT

The following **Operation** code requires a control structure of the following type:

- REFS_STREAM_SNAPSHOT_OPERATION_QUERY_DELTAS requires a REFS_STREAM_SNAPSHOT_QUERY_DELTAS_INPUT_BUFFER to be present.

2.3.65.1 REFS_STREAM_SNAPSHOT_QUERY_DELTAS_INPUT_BUFFER

Article04/06/2021

The REFS_STREAM_SNAPSHOT_QUERY_DELTAS_INPUT_BUFFER 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
StartingVcn																															
...																															
Flags																															
Reserved																															

StartingVcn (8 bytes): A signed integer representing the starting VCN for which to perform the request on.

Flags (4 bytes): An unsigned integer representing flags to modify the behavior of the request. This field must be set to zero.

Reserved (4 bytes): This field MUST be set to zero and MUST be ignored.

2.3.66 FSCTL_REFS_STREAM_SNAPSHOT_MANAGEMENT Reply

Article04/06/2021

This message returns the result of the FSCTL_REFS_STREAM_SNAPSHOT_MANAGEMENT request.

The message returns either a status code, as specified in section 2.2, or depending on the operation, an output data payload.

The most common error codes are listed in the following table.

Value	Meaning
STATUS_NOT_SUPPORTED 0xC00000BB	The operation as requested is not supported, or the file system does not support snapshot operations.
STATUS_INVALID_PARAMETER 0xC000000D	One of the parameters to the request is incorrect.
STATUS_INSUFFICIENT_RESOURCES 0xC000009A	There were insufficient resources to complete the operation.
STATUS_DISK_FULL 0xC000007F	The disk is full.
STATUS_MEDIA_WRITE_PROTECTED 0xC00000A2	The volume is read-only.
STATUS_SUCCESS 0x00000000	The operation was successful.

2.3.66.1 REFS_STREAM_SNAPSHOT_LIST_OUTPUT_BUFFER

Article04/06/2021

The REFS_STREAM_SNAPSHOT_LIST_OUTPUT_BUFFER 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
EntryCount																															
BufferSizeRequiredForQuery																															
Reserved																															
...																															
Entries (variable)																															
...																															

EntryCount (4 bytes): An unsigned integer representing the number of entries contained within the Entries field.

BufferSizeRequiredForQuery (4 bytes): An unsigned integer representing the total number of bytes to fully satisfy the request. This value is accurate upon returning STATUS_SUCCESS as well as STATUS_BUFFER_OVERFLOW.

Reserved (8 bytes): This field MUST be set to zero and MUST be ignored.

Entries (variable): An array of REFS_STREAM_SNAPSHOT_LIST_OUTPUT_BUFFER_ENTRY structs.

2.3.66.1.1 REFS_STREAM_SNAPSHOT_LIST_OUTPUT_BUFFER_ENTRY

Article04/06/2021

The REFS_STREAM_SNAPSHOT_LIST_OUTPUT_BUFFER_ENTRY 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																															
SnapshotNameLength																SnapshotCreationTime															
...																															
...																StreamSize															
...																															
...																StreamAllocationSize															
...																															
...																Reserved															
...																															
...																															
...																															
...																SnapshotName (variable)															
...																															

NextEntryOffset (4 bytes): An unsigned integer representing the offset in bytes to the next REFS_STREAM_SNAPSHOT_LIST_OUTPUT_BUFFER_ENTRY structure. When this value is zero there are no more entries in the array.

SnapshotNameLength (2 bytes): A unsigned integer representing the length of the UNICODE name contained in **SnapshotName** in bytes.

SnapshotCreationTime (8 bytes): An unsigned integer representing a FILETIME structure containing the creation time of the snapshot.

StreamSize (8 bytes): An unsigned integer representing the End-Of-File marker of the data stream represented by this entry.

StreamAllocationSize (8 bytes): An unsigned integer representing the size in bytes used by the data owned by the data stream represented by this entry.

Reserved (16 bytes): This field MUST be set to zero and MUST be ignored.

SnapshotName (variable): An array of WCHARs, as specified in [MS-DTYP] section 2.2.60, representing the UNICODE name for the snapshot representing this entry. The size of the array is defined in the **SnapshotNameLength** field.

2.3.66.2 REFS_STREAM_SNAPSHOT_QUERY_DELTAS_OUTPUT_BUFFER

Article04/06/2021

The REFS_STREAM_SNAPSHOT_QUERY_DELTAS_INPUT_BUFFER 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																															
Reserved																															
...																															
Extents (variable)																															
...																															

ExtentCount (4 bytes): An unsigned integer representing the number of REFS_STREAM_EXTENT structs contained in the Extents field.

Reserved (8 bytes): This field MUST be set to zero and MUST be ignored.

Extents (variable): An array of REFS_STREAM_EXTENT structs.

2.3.66.2.1 REFS_STREAM_EXTENT

Article04/06/2021

The REFS_STREAM_EXTENT 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
Vcn																															
...																															
Lcn																															
...																															
Length																															
...																															
Properties																															

Vcn (8 bytes): A signed integer representing a VCN within a data stream. This value will always be greater than zero.

Lcn (8 bytes): A signed integer representing the LCN mapping to Vcn in a data stream. This value will always be greater than zero.

Length (8 bytes): A signed integer representing the contiguous length in clusters for which the VCN to LCN mapping holds. This value will always be greater than zero.

Properties (4 bytes): A value representing the properties for this VCN to LCN mapping. The value MUST be one of the following:

Value	Meaning
REFS_STREAM_EXTENT_PROPERTY_VALID 0x0010	The metadata extent is considered valid, where the VCN to LCN mapping represents a written or zeroed extent.
REFS_STREAM_EXTENT_PROPERTY_STREAM_RESERVED 0x0020	The metadata extent does not map to an LCN, but instead contains a token representation an allocation reservation.
REFS_STREAM_EXTENT_PROPERTY_CRC32 0x0080	The metadata extent references data that is checksumed with the CRC32 algorithm.
REFS_STREAM_EXTENT_PROPERTY_CRC64 0x0100	The metadata extent references data that is checksumed with the CRC64 algorithm.
REFS_STREAM_EXTENT_PROPERTY_GHOSTED 0x0200	The metadata extent contains a ghosted recall buffer.

Value	Meaning
REFS_STREAM_EXTENT_PROPERTY_READONLY 0x0400	The metadata extent is a cached copy of a different metadata extent. This extent is immutable, and the LCN it references is not writable via this extent.
REFS_STREAM_EXTENT_PROPERTY_SPARSE 0x0008	The metadata extent represents a sparse range within the stream. The range represented by this extent is analogous to a sparse hole in the stream table.

2.3.67 FSCTL_SET_COMPRESSION Request

Article04/06/2021

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

The CompressionState element is as follows.

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
CompressionState																															

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. <62>
COMPRESSION_FORMAT_LZNT1 0x0002	The file or directory is compressed by using the LZNT1 compression algorithm. For more information, see [UASDC].
All other values	Reserved for future use and MUST NOT be used.

The actual file or directory compression performed when a server receives a request for COMPRESSION_FORMAT_DEFAULT and COMPRESSION_FORMAT_LZNT1 is implementation-dependent. <63>

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

2.3.68 FSCTL_SET_COMPRESSION Reply

Article04/07/2025

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 section 2.2. Upon success, the status code returned by the function that processes this FSCTL is STATUS_SUCCESS. The most common error codes are listed in the following table.

Expand table

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 one of the values listed in the table for **CompressionState** in FSCTL_SET_COMPRESSION Request (section 2.3.67).
STATUS_INVALID_DEVICE_REQUEST 0xC0000010	The volume does not allow compression.
STATUS_DISK_FULL 0xC00007F	The disk is full.

2.3.69 FSCTL_SET_DEFECT_MANAGEMENT Request

Article04/06/2021

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

This message contains a **FILE_SET_DEFECT_MGMT_BUFFER** structure.

FILE_SET_DEFECT_MGMT_BUFFER is defined as follows.

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
Disable																															

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

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

2.3.70 FSCTL_SET_DEFECT_MANAGEMENT Reply

Article04/07/2025

The only data item this message returns is a status code, as specified in section 2.2. Upon success, the status code returned directly by the function that processes this FSCTL is STATUS_SUCCESS. The most common error codes are listed in the following table.

Expand table

Error code	Meaning
STATUS_INVALID_PARAMETER 0xC000000D	An invalid parameter was passed to a service or function or the handle on which this FSCTL was invoked is that of a directory.
STATUS_INVALID_DEVICE_REQUEST 0xC0000010	The specified request is not a valid operation for the target device.
STATUS_SHARING_VIOLATION 0xC0000043	A file cannot be opened because the share access flags are incompatible.
STATUS_VOLUME_DISMOUNTED 0xC000026E	An operation was attempted to a volume after it was dismounted.
STATUS_FILE_INVALID 0xC0000098	The volume for a file has been externally altered such that the opened file is no longer valid.
STATUS_WRONG_VOLUME 0xC0000012	The wrong volume is in the drive.
STATUS_VERIFY_REQUIRED 0x80000016	The media has changed and a verify operation is in progress so no reads or writes can be performed to the device, except those used in the verify operation.

There are no additional data elements in this reply.

2.3.71 FSCTL_SET_ENCRYPTION Request

Article04/27/2022

The FSCTL_SET_ENCRYPTION request sets the encryption for the file or directory associated with the given handle.<64><65>

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

ENCRYPTION_BUFFER is defined as follows.

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
EncryptionOperation																															
Private								Padding																							

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

Value	Meaning
FILE_SET_ENCRYPTION 0x00000001	This operation requests encryption of the specified file or directory.<66>
FILE_CLEAR_ENCRYPTION 0x00000002	This operation requests removal of encryption from the specified file or directory. It MUST fail if any streams for the file are marked encrypted.<67>
STREAM_SET_ENCRYPTION 0x00000003	This operation requests encryption of the specified stream.<68>
STREAM_CLEAR_ENCRYPTION 0x00000004	This operation requests the removal of encryption from the specified stream.<69>

Private (1 byte): An 8-bit unsigned char value.<70>

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

2.3.72 FSCTL_SET_ENCRYPTION Reply

Article04/07/2025

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

This message returns a status code, as specified in section 2.2, as well as a DECRYPTION_STATUS_BUFFER (section 2.3.72.1) if an output buffer is passed in.

Upon success, the status code returned by the function that processes this FSCTL is STATUS_SUCCESS<71>. The most common error codes are listed in the following table.

Expand table

Error code	Meaning
STATUS_MEDIA_WRITE_PROTECTED 0xC00000A2	The disk cannot be written to because it is write-protected.
STATUS_INVALID_PARAMETER 0xC000000D	The **EncryptionOperation** field value is invalid, the open request is not for a file or directory or stream encryption has been requested on a stream that is compressed.
STATUS_BUFFER_TOO_SMALL 0xC0000023	The size of the input buffer is less than the size of the encryption buffer structure defined in section 2.3.71, or an output buffer is present and is smaller than a DECRYPTION_STATUS_BUFFER structure.
STATUS_VOLUME_NOT_UPGRADED 0xC000029C	The version of the file system on the volume does not support encryption.<72>
STATUS_INVALID_DEVICE_REQUEST 0xC0000010	The request was invalid for a system-specific reason.<73>
STATUS_FILE_CORRUPT_ERROR 0xC0000102	A required attribute is missing from a directory for which encryption was requested.<74>
STATUS_VOLUME_DISMOUNTED 0xC000026E	The volume is not mounted.
STATUS_INVALID_USER_BUFFER 0xC00000E8	An exception was raised while accessing a user buffer.

2.3.72.1 DECRYPTION_STATUS_BUFFER

Article04/07/2025

The **DECRYPTION_STATUS_BUFFER** is defined as follows.

Expand table

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
NoEncryptedStreams																															

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

2.3.73 FSCTL_SET_INTEGRITY_INFORMATION Request

Article04/27/2022

The FSCTL_SET_INTEGRITY_INFORMATION Request message requests that the server set the integrity state of the file or directory associated with the handle on which this FSCTL was invoked.<75>

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

The FSCTL_SET_INTEGRITY_INFORMATION_BUFFER 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
ChecksumAlgorithm																Reserved															
Flags																															

ChecksumAlgorithm (2 bytes): For **ReFS v1**, the field MUST be set to one of the following standard values.

Value	Meaning
CHECKSUM_TYPE_NONE 0x0000	The file or directory is set to not use integrity.
CHECKSUM_TYPE_CRC64 0x0002	The file or directory is set to provide integrity using a CRC64 checksum.
CHECKSUM_TYPE_UNCHANGED 0xFFFF	The integrity status of the file or directory is unchanged.
All other values 0x0003 — 0xFFFE	Reserved for future use and MUST NOT be used.

For **ReFS v2**, the field MUST be set to one of the following standard values.

Value	Meaning
CHECKSUM_TYPE_NONE 0x0000	The file or directory is set to not use integrity.
CHECKSUM_TYPE_CRC32 0x0001	The file or directory is set to provide integrity using a CRC32 or CRC64 checksum. If the ReFS cluster size is 4KB, the checksum used is CRC32; otherwise, if the cluster size is 64K, the CRC64 checksum is used.
CHECKSUM_TYPE_CRC64 0x0002	The file or directory is set to provide integrity using a CRC32 or CRC64 checksum. If the ReFS cluster size is 4KB, the checksum used is CRC32; otherwise, if the cluster size is 64K, the CRC64 checksum is used.

Value	Meaning
CHECKSUM_TYPE_UNCHANGED 0xFFFF	The integrity status of the file or directory is unchanged.
All other values 0x0003 — 0xFFFE	Reserved for future use and MUST NOT be used.

Note that for **ReFS v2** any value except CHECKSUM_TYPE_NONE or CHECKSUM_TYPE_UNCHANGED will set the integrity value to a file-system-selected integrity mechanism and is not guaranteed to use the user specified checksum value.

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

Flags (4 bytes): A 32-bit unsigned integer that contains zero or more of the following flag values. Flag values that are unspecified in the following table SHOULD be set to 0 and MUST be ignored.

Value	Meaning
FSCTL_INTEGRITY_FLAG_CHECKSUM_ENFORCEMENT_OFF 0x00000001	When set, if a checksum does not match, the associated I/O operation will not be failed.

2.3.74 FSCTL_SET_INTEGRITY_INFORMATION Reply

Article04/07/2025

This message returns the results of the FSCTL_SET_INTEGRITY_INFORMATION Request (section 2.3.73).

The only data item this message returns is a status code, as specified in section 2.2. Upon success, the status code returned by the function that processes this FSCTL is STATUS_SUCCESS. The most common error codes are listed in the following table.

Expand table

Error code	Meaning
STATUS_INVALID_PARAMETER 0xC000000D	The input buffer length is less than the size, in bytes, of the FSCTL_SET_INTEGRITY_INFORMATION_BUFFER element; the handle is not to a file or directory; or the requested **ChecksumAlgorithm** field is not one of the values listed in the table for the **ChecksumAlgorithm** field in the FSCTL_SET_INTEGRITY_INFORMATION Request.
STATUS_INVALID_DEVICE_REQUEST 0xC0000010	The volume does not support integrity.
STATUS_DISK_FULL 0xC000007F	The disk is full.
STATUS_NOT_SUPPORTED 0xC00000BB	The file has been ghosted (allocation blocks are being shared).

2.3.75 FSCTL_SET_INTEGRITY_INFORMATION_EX Request

Article04/27/2022

The FSCTL_SET_INTEGRITY_INFORMATION_EX Request message requests that the server set the integrity state of the file or directory associated with the handle on which this FSCTL was invoked.<76>

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

The FSCTL_SET_INTEGRITY_INFORMATION_BUFFER_EX 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
EnableIntegrity								A								Reserved															
Flags																															
Version								Reserved																							
...																															

EnableIntegrity (1 byte): This field MUST be one of the following values:

Value	Meaning
0x00	The file or directory is set to not use integrity.
0x01	The file or directory is set to provide integrity using CRC32 or CRC64 checksum.

A - KeepIntegrityStateUnchanged (1 byte): This field MUST be one of the following values:

Value	Meaning
0x00	The file or directory integrity state should change based on the EnableIntegrity parameter.
0x01	The file or directory integrity state must not change.

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

Flags (4 bytes): A 32-bit unsigned integer that contains zero or more of the following flag values. Flag values that are unspecified in the following table SHOULD be set to 0 and MUST be ignored.

Value	Meaning
FSCTL_INTEGRITY_FLAG_CHECKSUM_ENFORCEMENT_OFF 0x00000001	When set, if a checksum does not match, the associated I/O operation will not be failed.

Version (1 byte): An 8-bit value. This field MUST be set to 1.

Reserved (7 bytes): A 56-bit reserved value. This field MUST be set to zero and MUST be ignored.

2.3.76 FSCTL_SET_INTEGRITY_INFORMATION_EX Reply

Article04/07/2025

This message returns the results of the FSCTL_SET_INTEGRITY_INFORMATION_EX Request (section 2.3.75).

The only data item this message returns is a status code, as specified in section 2.2. Upon success, the status code returned by the function that processes this **FSCTL** is STATUS_SUCCESS. The most common error codes are listed in the following table.

Expand table

Error code	Meaning
STATUS_INVALID_PARAMETER 0xC000000D	The input buffer length is less than the size, in bytes, of the FSCTL_SET_INTEGRITY_INFORMATION_BUFFER_EX element; the handle is not to a file or directory; or Version is not equal to 1.
STATUS_INVALID_DEVICE_REQUEST 0xC0000010	The volume does not support integrity.
STATUS_DISK_FULL 0xC000007F	The disk is full.
STATUS_NOT_SUPPORTED 0xC00000BB	The file has been ghosted (allocation blocks are being shared).

2.3.77 FSCTL_SET_OBJECT_ID Request

Article04/07/2025

This message sets the object identifier for the file or directory associated with the handle on which this FSCTL was invoked. The message contains a FILE_OBJECTID_BUFFER (section 2.1.3) data element. Either a Type 1 or a Type 2 buffer is valid.<77><78>

2.3.78 FSCTL_SET_OBJECT_ID Reply

Article04/07/2025

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 section 2.2. Upon success, the status code returned by the function that processes this FSCTL is STATUS_SUCCESS. The most common error codes are listed in the following table.

Expand table

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

2.3.79 FSCTL_SET_OBJECT_ID_EXTENDED Request

Article04/07/2025

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.

Expand table

0	1	2	3	4	5	6	7	8	9	10	1	2	3	4	5	6	7	8	9	20	1	2	3	4	5	6	7	8	9	30	1
ExtendedInfo (48 bytes)																															
...																															
...																															

ExtendedInfo (48 bytes): A 48-byte binary large object(BLOB) containing user-defined extended data that was passed to this FSCTL by an application. In this situation, the user refers to the implementer who is calling this FSCTL, meaning the extended info is opaque to NTFS; there are no rules enforced by NTFS as to what these last 48 bytes contain. Contrast this with the first 16 bytes of an object ID, which can be used to open the file, so NTFS requires that they be unique within a volume.<79>

2.3.80 FSCTL_SET_OBJECT_ID_EXTENDED Reply

Article04/07/2025

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 section 2.2. Upon success, the status code returned by the function that processes this FSCTL is STATUS_SUCCESS. The most common error codes are listed in the following table.

Expand table

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.

2.3.81 FSCTL_SET_REPARSE_POINT Request

Article04/07/2025

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

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

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

2.3.82 FSCTL_SET_REPARSE_POINT Reply

06/10/2025

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 section 2.2. Upon success, the status code returned by the function that processes this FSCTL is STATUS_SUCCESS. The most common error codes are listed in the following table.

Expand table

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

2.3.83 FSCTL_SET_SPARSE Request

Article04/07/2025

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) might not require disk allocation. Space for nonzero data is allocated as the file is written. The message either has no data elements at all or it contains a FILE_SET_SPARSE_BUFFER element. If there is no data element, the sparse flag for the file is set, exactly as if the FILE_SET_SPARSE_BUFFER element was supplied and had a **SetSparse** value of TRUE.<80>

The **FILE_SET_SPARSE_BUFFER** element is as follows:

Expand table

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
SetSparse																															

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

A FALSE value will cause the file system to attempt to "unsparse" the file by allocating clusters for any regions of the file that are currently sparsed. If the entire file is successfully unsparsed, the sparse flag is cleared for the file. If an error is encountered during unsparsing, any regions of the file that were unsparsed MAY<81> remain unsparsed.

A TRUE value will cause the sparse flag for the file to set. Currently allocated clusters SHOULD NOT<82> be deallocated.

2.3.84 FSCTL_SET_SPARSE Reply

Article04/07/2025

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 section 2.2. Upon success, the status code returned by the function that processes this FSCTL is STATUS_SUCCESS. The most common error codes are listed in the following table.

Expand table

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.

2.3.85 FSCTL_SET_ZERO_DATA Request

06/10/2025

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.

Expand table

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
FileOffset																															
...																															
BeyondFinalZero																															
...																															

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.<83>

2.3.86 FSCTL_SET_ZERO_DATA Reply

Article04/07/2025

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 section 2.2. Upon success, the status code returned by the function that processes this FSCTL is STATUS_SUCCESS. The most common error codes are listed in the following table.

Expand table

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.

2.3.87 FSCTL_SET_ZERO_ON_DEALLOCATION Request

Article04/06/2021

This message requests that the server fill the clusters of the target file with zeros when they are deallocated.<84> This is used to set a file to secure delete mode, which ensures that data will be zeroed upon file truncation or deletion.

There are several side effects associated with this operation.

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

This message does not contain any additional data elements.

2.3.88 FSCTL_SET_ZERO_ON_DEALLOCATION Reply

Article04/07/2025

This message returns the results of the FSCTL_SET_ZERO_ON_DEALLOCATION request. The only data item this message returns is a status code, as specified in section 2.2. Upon success, the status code returned by the function that processes this FSCTL is STATUS_SUCCESS. The most common error codes are listed in the following table.

Expand table

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

2.3.89 FSCTL_SIS_COPYFILE Request

Article04/27/2022

The FSCTL_SIS_COPYFILE request message requests that the server use the single-instance storage (SIS) filter to copy a file. The message contains an SI_COPYFILE data element.

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

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

The SI_COPYFILE data element is as follows.

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
SourceFileNameLength																															
DestinationFileNameLength																															
Flags																															
SourceFileName (variable)																															
...																															
DestinationFileName (variable)																															
...																															

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

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

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

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

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

DestinationFileName (variable): A null-terminated Unicode string containing the destination file name. <85>

2.3.90 FSCTL_SIS_COPYFILE Reply

Article04/07/2025

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 section 2.2. Upon success, the status code returned by the function that processes this FSCTL is STATUS_SUCCESS. The most common error codes are listed in the following table.

Expand table

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

Error code	Meaning
0xC0000022	

2.3.91 FSCTL_VIRTUAL_STORAGE_QUERY_PROPERTY Request

Article04/07/2025

This request contains a message with the same structure as the IOCTL_STORAGE_QUERY_PROPERTY request (section 2.8.1) with the following values:

PropertyId (4 bytes): 0x00000004

QueryType (4 bytes): 0x00000000

Remote servers SHOULD ignore this request.<86>

2.3.92 FSCTL_WRITE_USN_CLOSE_RECORD Request

Article04/07/2025

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

No data structure is associated with this request.

2.3.93 FSCTL_WRITE_USN_CLOSE_RECORD Reply

Article04/07/2025

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 for the file or directory. This value MUST be greater than or equal to 0.

This message returns a status code as specified in section 2.2. Upon success, the status code returned by the function that processes this FSCTL is STATUS_SUCCESS. The most common error codes are listed in the following table.

Expand table

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

2.4 File Information Classes

File information classes are numerical values (specified by the Level column in the following table) that specify what information for a file is to be queried or set or for local use<87>. File information classes can require additional information to be included in the query or the response. When appropriate, the additional information is detailed in the file information class description. The table indicates which file information classes are supported for query and set operations.<88>

Expand table

File information class	Level	Uses
FileAccessInformation	8	Query
FileAlignmentInformation	17	Query
FileAllInformation	18	Query
FileAllocationInformation	19	Set
FileAlternateNameInformation	21	Query
FileAttributeTagInformation	35	Query
FileBasicInformation	4	Query, Set
FileBothDirectoryInformation	3	Query
FileCompressionInformation	28	Query
FileDirectoryInformation	1	Query
FileDispositionInformation	13	Set
FileDispositionInformationEx	64	Set<89>
FileEaInformation	7	Query
FileEndOfFileInformation	20	Set
FileFullDirectoryInformation	2	Query
FileFullEaInformation	15	Query, Set
FileHardLinkInformation	46	LOCAL
FileId64ExtdBothDirectoryInformation	79	Query<90>
FileId64ExtdDirectoryInformation	78	Query<91>

File information class	Level	Uses
FileIdAllExtdBothDirectoryInformation	81	Query<92>
FileIdAllExtdDirectoryInformation	80	Query<93>
FileIdBothDirectoryInformation	37	Query
FileIdExtdDirectoryInformation	60	Query
FileIdFullDirectoryInformation	38	Query
FileIdGlobalTxDirectoryInformation	50	LOCAL
FileIdInformation	59	Query<94>
FileInternalInformation	6	Query
FileLinkInformation	11	Set
FileMailslotQueryInformation	26	LOCAL
FileMailslotSetInformation	27	LOCAL
FileModeInformation	16	Query, Set<95>
FileMoveClusterInformation	31	<96>
FileNameInformation	9	LOCAL
FileNamesInformation	12	Query
FileNetworkOpenInformation	34	Query
FileNormalizedNameInformation	48	Query<97>
FileObjectIdInformation	29	LOCAL
FilePipeInformation	23	Query, Set
FilePipeLocalInformation	24	Query
FilePipeRemoteInformation	25	Query
FilePositionInformation	14	Query, Set
FileQuotaInformation	32	Query, Set<98>
FileRenameInformation	10	Set
FileRenameInformationEx	65	Set<99>
FileReparsePointInformation	33	LOCAL

File information class	Level	Uses
FileSfioReserveInformation	44	LOCAL
FileSfioVolumeInformation	45	<100>
FileShortNameInformation	40	Set
FileStandardInformation	5	Query
FileStandardLinkInformation	54	LOCAL
FileStreamInformation	22	Query
FileTrackingInformation	36	LOCAL
FileValidDataLengthInformation	39	Set

If an information class is specified that does not match the usage in the above table, STATUS_INVALID_INFO_CLASS MUST be returned. If a file system does not support a specific file information class, STATUS_INVALID_PARAMETER MUST be returned.

Last updated on 11/21/2025

2.4.1 FileAccessInformation

Article04/07/2025

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

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

Expand table

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

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

This operation returns a status code as specified in section 2.2. Upon success, the status code returned by the function that processes this file information class is STATUS_SUCCESS. The most common error codes are listed in the following table.

Expand table

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

2.4.2 FileAllInformation

Article05/28/2021

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

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

0	1	2	3	4	5	6	7	8	9	10	1	2	3	4	5	6	7	8	9	20	1	2	3	4	5	6	7	8	9	30	1
BasicInformation (40 bytes)																															
...																															
...																															
StandardInformation (24 bytes)																															
...																															
...																															
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 2.4.41.

InternalInformation (8 bytes): A FILE_INTERNAL_INFORMATION structure specified in section 2.4.22.

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 2.4.35.

ModeInformation (4 bytes): A FILE_MODE_INFORMATION structure specified in section 2.4.26.

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

NameInformation (variable): A FILE_NAME_INFORMATION structure specified in section 2.4.27.

This operation returns a status code as specified in section 2.2. Upon success, the status code returned by the function that processes this file information class is STATUS_SUCCESS. The most common error codes are listed in the following table.

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

2.4.3 FileAlignmentInformation

Article04/07/2025

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

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

Expand table

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
AlignmentRequirement																															

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

Expand table

Value	Meaning
FILE_BYTE_ALIGNMENT 0x00000000	Specifies that there are no alignment requirements for the device.
FILE_WORD_ALIGNMENT 0x00000001	Specifies that data MUST be aligned on a 2-byte boundary.
FILE_LONG_ALIGNMENT 0x00000003	Specifies that data MUST be aligned on a 4-byte boundary.
FILE_QUAD_ALIGNMENT 0x00000007	Specifies that data MUST be aligned on an 8-byte boundary.
FILE_OCTA_ALIGNMENT 0X0000000F	Specifies that data MUST be aligned on a 16-byte boundary.
FILE_32_BYTE_ALIGNMENT 0X0000001F	Specifies that data MUST be aligned on a 32-byte boundary.
FILE_64_BYTE_ALIGNMENT 0X0000003F	Specifies that data MUST be aligned on a 64-byte boundary.
FILE_128_BYTE_ALIGNMENT 0X0000007F	Specifies that data MUST be aligned on a 128-byte boundary.
FILE_256_BYTE_ALIGNMENT 0X000000FF	Specifies that data MUST be aligned on a 256-byte boundary.

Value	Meaning
FILE_512_BYTE_ALIGNMENT 0X000001FF	Specifies that data MUST be aligned on a 512-byte boundary.

This operation returns a status code as specified in section 2.2. Upon success, the status code returned by the function that processes this file information class is STATUS_SUCCESS. The most common error codes are listed in the following table.

Expand table

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

2.4.4 FileAllocationInformation

Article04/07/2025

This information class is used to set but not to query the allocation size for a file. The file system is passed a 64-bit signed integer containing the file allocation size, in bytes. The file system rounds the requested allocation size up to an integer multiple of the cluster size for nonresident files, or an implementation-defined multiple for resident files.<101><102> All unused allocation (beyond EOF) is freed on the last handle close.

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

Expand table

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
AllocationSize																															
...																															

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

This operation returns a status code as specified in section 2.2. Upon success, the status code returned by the function that processes this file information class is STATUS_SUCCESS. The most common error codes are listed in the following table.

Expand table

Error code	Meaning
STATUS_INVALID_PARAMETER 0xC000000D	The handle is for a directory and not a file, or the allocation is greater than the maximum file size allowed.
STATUS_ACCESS_DENIED 0xC0000022	The handle was not opened to write file data or file attributes.
STATUS_DISK_FULL 0xC000007F	The disk is full.
STATUS_INFO_LENGTH_MISMATCH 0xC0000004	The specified information record length does not match the length that is required for the specified information class.

2.4.5 FileAlternateNameInformation

Article04/07/2025

This information class is used to query alternate name information for a file. The alternate name for a file is its 8.3 format name (eight characters that appear before the "." and three characters that appear after). A file MAY have an alternate name to achieve compatibility with the 8.3 naming requirements of legacy applications.<103>

A FILE_NAME_INFORMATION (section 2.1.7) data element containing an 8.3 file name (section 2.1.5.2.1) is returned by the server.

This operation returns a status code as specified in section 2.2. Upon success, the status code returned by the function that processes this file information class is STATUS_SUCCESS. The most common error codes are listed in the following table.

Expand table

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

2.4.6 FileAttributeTagInformation

Article04/07/2025

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

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

Expand table

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
FileAttributes																															
ReparseTag																															

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

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

This operation returns a status code as specified in section 2.2. Upon success, the status code returned by the function that processes this file information class is STATUS_SUCCESS. The most common error codes are listed in the following table.

Expand table

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

2.4.7 FileBasicInformation

Article02/10/2025

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

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

Expand table

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
CreationTime																															
...																															
LastAccessTime																															
...																															
LastWriteTime																															
...																															
ChangeTime																															
...																															
FileAttributes																															
Reserved																															

CreationTime (8 bytes): The time when the file was created; see section 2.1.1. A valid time for this field is an integer greater than or equal to 0. When setting file attributes, a value of 0 indicates to the server that it MUST NOT change this attribute. When setting file attributes, a value of -1 indicates to the server that it MUST NOT change this attribute for all subsequent operations on the same file handle. When setting file attributes, a value of -2 indicates to the server that it MUST change this attribute for all subsequent operations on the same file handle. This field MUST NOT be set to a value less than -2. <102>

LastAccessTime (8 bytes): The last time the file was accessed; see section 2.1.1. A valid time for this field is an integer greater than or equal to 0. When setting file attributes, a value of 0 indicates to the server that it MUST NOT change this attribute. When setting file attributes, a value of -1 indicates to the server that it MUST NOT change this attribute for all subsequent operations on the same file handle. When setting file attributes, a value of -2 indicates to the server that it MUST change this attribute for all subsequent operations on the same file handle. This field MUST NOT be set to a value less than -2. <103>

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

When setting file attributes, a value of -2 indicates to the server that it MUST change this attribute for all subsequent operations on the same file handle. This field MUST NOT be set to a value less than -2. <104>

ChangeTime (8 bytes): The last time the file was changed; see section 2.1.1. A valid time for this field is an integer greater than or equal to 0. When setting file attributes, a value of 0 indicates to the server that it MUST NOT change this attribute. When setting file attributes, a value of -1 indicates to the server that it MUST NOT change this attribute for all subsequent operations on the same file handle. When setting file attributes, a value of -2 indicates to the server that it MUST change this attribute for all subsequent operations on the same file handle. This field MUST NOT be set to a value less than -2. <105>

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

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

This operation returns a status code as specified in section 2.2. Upon success, the status code returned by the function that processes this file information class is STATUS_SUCCESS. The most common error codes are listed in the following table.

Expand table

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

2.4.8 FileBothDirectoryInformation

Article12/14/2021

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

This information class returns a list that contains a **FILE_BOTH_DIR_INFORMATION** data element for each file or directory within the target directory.

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

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

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

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 (24 bytes)									
...																															

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

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

FileIndex (4 bytes): A 32-bit unsigned integer that contains the byte offset of the file within the parent directory. For file systems in which the position of a file within the parent directory is not fixed and can be changed at any time to maintain sort order, this field SHOULD be set to 0 and MUST be ignored. <101>

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

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

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

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

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

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

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

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

EaSize (4 bytes): If **FILE_ATTRIBUTE_REPARSE_POINT** is set in the **FileAttributes** field, this field MUST contain a reparse tag as specified in section 2.1.2.1. Otherwise, this field is a 32-bit unsigned integer that contains the combined length, in bytes, of the extended attributes (EA) for the file.

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

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

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

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

This operation returns a status code as specified in section 2.2. Upon success, the status code returned by the function that processes this file information class is STATUS_SUCCESS. The most common error codes are listed in the following table.

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

2.4.9 FileCompressionInformation

Article05/28/2021

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

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

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
CompressedFileSize																															
...																															
CompressionFormat																CompressionUnitShift								ChunkShift						
ClusterShift								Reserved																							

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

CompressionFormat (2 bytes): A 16-bit unsigned integer that contains the compression format. The actual compression operation associated with each of these compression format values is implementation-dependent. An implementation can link any local compression algorithm with the values described in the following table because the compressed data does not travel across the wire in the context of FSCTL, FileInformation class, or FileSystemInformation class requests or replies.<102>

Value	Meaning
COMPRESSION_FORMAT_NONE 0x0000	The file or directory is not compressed.
COMPRESSION_FORMAT_LZNT1 0x0002	The file or directory is compressed by using the LZNT1 compression algorithm.
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.<103>

ChunkShift (1 byte): An 8-bit unsigned integer that contains the compression chunk size shift, which is the number of bits by which to left-shift a 1 bit to arrive at the compression chunk size. The chunk size is the number of bytes that the operating system's implementation of the Lempel-Ziv compression algorithm tries to compress at one time. This value is implementation-defined.<104>

ClusterShift (1 byte): An 8-bit unsigned integer that contains the cluster size shift, which is the number of bits by which to left-shift a 1 bit to arrive at the cluster size. The cluster size specifies the amount of space that is saved by compression to successfully compress a compression unit. If a cluster size amount of space is not saved by compression, the data in that compression unit is stored uncompressed. Each

successfully compressed compression unit MUST occupy at least one cluster less than the uncompressed compression unit. This value is implementation-defined.<105>

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

This operation returns a status code as specified in section 2.2. Upon success, the status code returned by the function that processes this file information class is STATUS_SUCCESS. The most common error codes are listed in the following table.

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

2.4.10 FileDirectoryInformation

Article04/27/2022

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

This information class returns a list that contains a **FILE_DIRECTORY_INFORMATION** data element for each file or directory within the target directory.

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

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

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

NextEntryOffset (4 bytes): A 32-bit unsigned integer that contains the byte offset from the beginning of this entry, at which the next FILE_DIRECTORY_INFORMATION entry is located, if multiple entries are

present in a buffer. This member MUST be zero if no other entries follow this one. An implementation MUST use this value to determine the location of the next entry (if multiple entries are present in a buffer).

FileIndex (4 bytes): A 32-bit unsigned integer that contains the byte offset of the file within the parent directory. For file systems in which the position of a file within the parent directory is not fixed and can be changed at any time to maintain sort order, this field SHOULD be set to 0 and MUST be ignored. <106>

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

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

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

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

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

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

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

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

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

This operation returns a status code as specified in section 2.2. Upon success, the status code returned by the function that processes this file information class is STATUS_SUCCESS. The most common error codes are listed in the following table.

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

2.4.11 FileDispositionInformation

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

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

Expand table

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
DeletePending																															

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 which means the file SHOULD NOT be deleted.<114>

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

This operation returns a status code as specified in section 2.2. Upon success, the status code returned by the function that processes this file information class is STATUS_SUCCESS. The most common error codes are listed in the following table.

Expand table

Error code	Meaning
STATUS_ACCESS_DENIED 0xC0000022	The handle was not opened with delete access.
STATUS_INFO_LENGTH_MISMATCH 0xC0000004	The specified information record length does not match the length that is required for the specified information class.
STATUS_DIRECTORY_NOT_EMPTY 0xC0000101	Indicates that the directory trying to be deleted is not empty.

Last updated on 11/21/2025

2.4.12 FileDispositionInformationEx

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

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

Expand table

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
Flags																															

Flags (4 bytes): A 32-bit field that specifies options on how the file is deleted.

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

Expand table

Value	Meaning
FILE_DISPOSITION_DO_NOT_DELETE_FILE 0x00000000	If no flag is set, the file MUST NOT be deleted.
FILE_DISPOSITION_DELETE 0x00000001	If set, indicates the file SHOULD be deleted.
FILE_DISPOSITION_POSIX_SEMANTICS 0x00000002	If set and FILE_DISPOSITION_DELETE is set, indicates the file SHOULD be deleted using POSIX-style semantics. This means the link is removed from the visible namespace as soon as the POSIX delete handle is closed, but the file's data streams remain accessible by other existing handles.
FILE_DISPOSITION_FORCE_IMAGE_SECTION_CHECK 0x00000004	If set, indicates the system SHOULD fail deleting the file if an image section exists. If not set and the FILE_DISPOSITION_POSIX_SEMANTICS flag is set; indicates the file can be deleted even if it has an image section. This flag was added to support backward compatibility with the existing behavior of the FileDispositionInformation (see section 2.4.11) operation.
FILE_DISPOSITION_ON_CLOSE 0x00000008	If set and the FILE_DISPOSITION_POSIX_SEMANTICS flag is set; the file FILE_DELETE_ON_CLOSE state is updated to specify POSIX-style delete semantics. If set and the FILE_DISPOSITION_POSIX_SEMANTICS flag is **not** set; the file FILE_DELETE_ON_CLOSE state is updated to **not** specify POSIX-style delete semantics. If set and the file is not opened with FILE_DELETE_ON_CLOSE, STATUS_NOT_SUPPORTED MUST be returned.
FILE_DISPOSITION_IGNORE_READONLY_ATTRIBUTE 0x00000010	If set, allows files with the READ_ONLY attribute to be deleted anyway. Without this flag, deleting a read-only file MUST return STATUS_CANNOT_DELETE.

This operation returns a status code as specified in section 2.2. Upon success, the status code returned by the function that processes this file information class is STATUS_SUCCESS. The most common error codes are listed in the following table.

Expand table

Error code	Meaning
STATUS_ACCESS_DENIED 0xC0000022	The handle was not opened with delete access.
STATUS_INFO_LENGTH_MISMATCH 0xC0000004	The specified information record length does not match the length that is required for the specified information class.
STATUS_DIRECTORY_NOT_EMPTY 0xC0000101	Indicates that the directory trying to be deleted is not empty.
STATUS_CANNOT_DELETE 0xC0000121	An attempt has been made to remove a file or directory that cannot be deleted.

Last updated on 11/21/2025

2.4.13 FileEaInformation

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

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

Expand table

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
EaSize																															

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

This operation returns a status code as specified in section 2.2. Upon success, the status code returned by the function that processes this file information class is STATUS_SUCCESS. The most common error codes are listed in the following table.

Expand table

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

Last updated on 11/21/2025

2.4.13 FileEndOfFileInformation

Article10/30/2020

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

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

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
EndOfFile																															
...																															

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

This operation returns a status code as specified in section 2.2. Upon success, the status code returned by the function that processes this file information class is STATUS_SUCCESS. The most common error codes are listed in the following table.

Error code	Meaning
STATUS_INVALID_PARAMETER 0xC000000D	The handle was for a directory and not a file, or the allocation is greater than the maximum file size allowed.
STATUS_ACCESS_DENIED 0xC0000022	The handle was not opened to read file data or file attributes.
STATUS_DISK_FULL 0xC000007F	The disk is full.
STATUS_INFO_LENGTH_MISMATCH 0xC0000004	The specified information record length does not match the length that is required for the specified information class.

2.4.14 FileFullDirectoryInformation

Article05/28/2021

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

This information class returns a list that contains a **FILE_FULL_DIR_INFORMATION** data element for each file or directory within the target directory.

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

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

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																															
FileName (variable)																															
...																															

NextEntryOffset (4 bytes): A 32-bit unsigned integer that contains the byte offset from the beginning of this entry, at which the next FILE_FULL_DIR_INFORMATION entry is located, if multiple entries are present

in a buffer. This member is zero if no other entries follow this one. An implementation MUST use this value to determine the location of the next entry (if multiple entries are present in a buffer).

FileIndex (4 bytes): A 32-bit unsigned integer that contains the byte offset of the file within the parent directory. For file systems 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. <108>

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

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

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

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

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

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

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

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

EaSize (4 bytes): If **FILE_ATTRIBUTE_REPARSE_POINT** is set in the **FileAttributes** field, this field MUST contain a reparse tag as specified in section 2.1.2.1. Otherwise, this field is a 32-bit unsigned integer that contains the combined length, in bytes, of the extended attributes (EA) for the file.

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

This operation returns a status code as specified in section 2.2. Upon success, the status code returned by the function that processes this file information class is STATUS_SUCCESS. The most common error codes are listed in the following table.

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

2.4.16 FileFullEaInformation

Article04/07/2025

This information class is used to query or set extended attribute (EA) information for a file. For queries, the client provides a list of FILE_GET_EA_INFORMATION (section 2.4.16.1) structures, and a list of **FILE_FULL_EA_INFORMATION** structures is returned by the server. For setting EA information, the client provides a list of **FILE_FULL_EA_INFORMATION** structures, and a status code is returned by the server, as specified in section 2.2.

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

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

Expand table

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																															
Flags								EaNameLength								EaValueLength															
EaName (variable)																															
...																															
EaValue (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_EA_INFORMATION entry is located, if multiple entries are present in the buffer. This member MUST be zero if no other entries follow this one. An implementation MUST use this value to determine the location of the next entry (if multiple entries are present in a buffer).

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

Expand table

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

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

EaName.

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

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

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

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

This operation returns a status code as specified in section 2.2. Upon success, the status code returned by the function that processes this file information class is STATUS_SUCCESS. The most common error codes are listed in the following table.

Expand table

Error code	Meaning
STATUS_INVALID_DEVICE_REQUEST 0xC0000010	The target file system does not implement this functionality.
STATUS_ACCESS_DENIED 0xC0000022	The handle was not opened to read file data or file attributes.
STATUS_BUFFER_TOO_SMALL 0xC0000023	The buffer is too small to contain the entry. No information has been written to the buffer.
STATUS_NO_EAS_ON_FILE 0xC0000052	The file for which EAs were requested has no EAs.
STATUS_BUFFER_OVERFLOW 0x80000005	The output buffer was filled before all of the EA data could be returned. Only complete FILE_FULL_EA_INFORMATION structures are returned.
STATUS_INVALID_EA_NAME 0x80000013	The **Flags** field contains a value other than zero or FILE_NEED_EA, or the **EaName** field is longer than 255 characters, or it contains any of the following characters: ASCII values 0x00 - 0x1F, \ / : * ? " < > | , + = [ ] ;

2.4.16.1 FILE_GET_EA_INFORMATION

Article04/07/2025

This data structure can be used to specify an explicit list of attributes to query via the FileFullEaInformation (section 2.4.16) information class. If no FILE_GET_EA_INFORMATION elements are specified, all extended attributes for the given file are returned.

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

Expand table

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																															
EaNameLength								EaName (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_GET_EA_INFORMATION** entry is located, if multiple entries are present in a buffer. This member MUST be zero if no other entries follow this one. An implementation MUST use this value to determine the location of the next entry (if multiple entries are present in a buffer).

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

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

2.4.17 FileHardLinkInformation

Article04/07/2025

This information class is used locally to query hard links to an existing file.<116> At least one name MUST be returned.

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

Expand table

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
BytesNeeded																															
EntriesReturned																															
Entries (variable)																															
...																															

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.

The query MUST return as many entries as will fit in the supplied output buffer. A value of 0x00000000 for this field indicates that there is insufficient room to return any entry. The error STATUS_BUFFER_OVERFLOW (0x80000005) indicates that not all available entries were returned.

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

This operation returns a status code as specified in section 2.2. Upon success, the status code returned by the function that processes this file information class is STATUS_SUCCESS. The most common error codes are listed in the following table.

Expand table

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

2.4.17.1 FILE_LINK_ENTRY_INFORMATION

Article04/07/2025

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

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

Expand table

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																															
ParentFileId																															
...																															
FileNameLength																															
FileName (variable)																															
...																															

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): The 64-bit file ID, as specified in section 2.1.9, of the parent directory of the given link. For file systems which do not support a 64-bit file ID, this field MUST be set to 0, and MUST be ignored.

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

FileName (variable): A sequence of **FileNameLength** Unicode characters that MUST contain the Unicode string name of the given link.

2.4.18 FileId64ExtdBothDirectoryInformation

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

This information class returns a list that contains a **FILE_ID_64_EXTD_BOTH_DIR_INFORMATION** data element for each file or directory within the target directory.

When multiple **FILE_ID_64_EXTD_BOTH_DIR_INFORMATION** data elements are present in the buffer, each MUST be aligned on an 8-byte boundary. Any bytes inserted for alignment SHOULD be set to zero, and the receiver MUST ignore them. No padding is required following the last data element.

A **FILE_ID_64_EXTD_BOTH_DIR_INFORMATION** data element is as follows.

Expand table

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																															
ReparsePointTag																															
FileId																															

...		
ShortNameLength	Reserved1	ShortName (24 bytes)
...		
...		
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_64_EXTD_BOTH_DIR_INFORMATION entry is located, if multiple entries are present in the buffer. This member MUST be zero if no other entries follow this one. An implementation MUST use this value to determine the location of the next entry (if multiple entries are present in a buffer).

FileIndex (4 bytes): A 32-bit unsigned integer that contains the byte offset of the file within the parent directory. For file systems in which the position of a file within the parent directory is not fixed and can be changed at any time to maintain sort order, this field SHOULD be set to 0 and MUST be ignored. <117>

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

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

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

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

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

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

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

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

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

ReparsePointTag (4 bytes): If **FILE_ATTRIBUTE_REPARSE_POINT** is set in the **FileAttributes** field, this field MUST contain a 32-bit unsigned integer value containing the reparse point tag that uniquely identifies the owner of the reparse point. Section 2.1.2.1 contains more details on reparse tags.

FileId (8 bytes): The 64-bit file ID, as specified in section 2.1.9, for the file. For file systems that do not support a 64-bit file ID, this field MUST be set to 0, and MUST be ignored. For file systems which do not explicitly store directory entries named ".." (synonymous with the parent directory), an implementation MAY set this field to 0 for the entry named "..", and this value MUST be ignored.<118>

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

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

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

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

This operation returns a status code as specified in section 2.2. Upon success, the status code returned by the function that processes this file information class is STATUS_SUCCESS. The most common error codes are listed in the following table.

Expand table

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

Last updated on 11/21/2025

2.4.18 FileId64ExtdDirectoryInformation

Article07/03/2024

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

This information class returns a list that contains a **FILE_ID_64_EXTD_DIR_INFORMATION** data element for each file or directory within the target directory.

When multiple **FILE_ID_64_EXTD_DIR_INFORMATION** data elements are present in the buffer, each MUST be aligned on an 8-byte boundary. Any bytes inserted for alignment SHOULD be set to zero, and the receiver MUST ignore them. No padding is required following the last data element.

A **FILE_ID_64_EXTD_DIR_INFORMATION** data element is as follows.

Expand table

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																															
ReparsePointTag																															
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_64_EXTD_DIR_INFORMATION entry is located, if multiple entries are present in the buffer. This member MUST be zero if no other entries follow this one. An implementation MUST use this value to determine the location of the next entry (if multiple entries are present in a buffer).

FileIndex (4 bytes): A 32-bit unsigned integer that contains the byte offset of the file within the parent directory. For file systems in which the position of a file within the parent directory is not fixed and can be changed at any time to maintain sort order, this field SHOULD be set to 0 and MUST be ignored. <117>

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

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

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

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

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

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

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

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

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

ReparsePointTag (4 bytes): If **FILE_ATTRIBUTE_REPARSE_POINT** is set in the **FileAttributes** field, this field MUST contain a 32-bit unsigned integer value containing the reparse point tag that uniquely identifies the owner of the reparse point. section 2.1.2.1 contains more details on reparse tags.

FileId (8 bytes): The 64-bit file ID, as specified in section 2.1.9, for the file. For file systems that do not support a 64-bit file ID, this field MUST be set to 0, and MUST be ignored. For file systems which do not explicitly store directory entries named ".." (synonymous with the parent directory), an implementation MAY set this field to 0 for the entry named "..", and this value MUST be ignored. <118>

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

This operation returns a status code as specified in section 2.2. Upon success, the status code returned by the function that processes this file information class is STATUS_SUCCESS. The most common error codes are listed in the following table.

Expand table

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

2.4.19 FileIdAllExtdBothDirectoryInformation

Article07/03/2024

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

This information class returns a list that contains a **FILE_ID_ALL_EXTD_BOTH_DIR_INFORMATION** data element for each file or directory within the target directory.

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

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

Expand table

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																															
ReparsePointTag																															
FileId																															

...		
FileId128		
...		
...		
...		
ShortNameLength	Reserved1	ShortName (24 bytes)
...		
...		
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_ALL_EXTD_BOTH_DIR_INFORMATION entry is located, if multiple entries are present in the buffer. This member MUST be zero if no other entries follow this one. An implementation MUST use this value to determine the location of the next entry (if multiple entries are present in a buffer).

FileIndex (4 bytes): A 32-bit unsigned integer that contains the byte offset of the file within the parent directory. For file systems in which the position of a file within the parent directory is not fixed and can be changed at any time to maintain sort order, this field SHOULD be set to 0 and MUST be ignored. <119>

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

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

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

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

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

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

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

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

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

ReparsePointTag (4 bytes): If **FILE_ATTRIBUTE_REPARSE_POINT** is set in the **FileAttributes** field, this field MUST contain a 32-bit unsigned integer value containing the reparse point tag that uniquely identifies the owner of the reparse point. section 2.1.2.1 contains more details on reparse tags.

FileId (8 bytes): The 64-bit file ID, as specified in section 2.1.9, for the file. For file systems that do not support a 64-bit file ID, this field MUST be set to 0, and MUST be ignored. For file systems which do not explicitly store directory entries named ".." (synonymous with the parent directory), an implementation MAY set this field to 0 for the entry named "..", and this value MUST be ignored. <120>

FileId128 (16 bytes): The 128-bit file ID, as specified in section 2.1.10, of the file. For file systems that do not support a 128-bit file ID, this field MUST be set to 0, and MUST be ignored.

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

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

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

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

This operation returns a status code as specified in section 2.2. Upon success, the status code returned by the function that processes this file information class is STATUS_SUCCESS. The most common error codes are listed in the following table.

Expand table

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

2.4.21 FileIdAllExtdDirectoryInformation

Article04/07/2025

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

This information class returns a list that contains a **FILE_ID_ALL_EXTD_DIR_INFORMATION** data element for each file or directory within the target directory.

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

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

Expand table

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																																
ReparsePointTag																																
FileId																																

...	
FileId128	
...	
...	
...	
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_ALL_EXTD_DIR_INFORMATION entry is located, if multiple entries are present in the buffer. This member MUST be zero if no other entries follow this one. An implementation MUST use this value to determine the location of the next entry (if multiple entries are present in a buffer).

FileIndex (4 bytes): A 32-bit unsigned integer that contains the byte offset of the file within the parent directory. For file systems in which the position of a file within the parent directory is not fixed and can be changed at any time to maintain sort order, this field SHOULD be set to 0 and MUST be ignored. <123>

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

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

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

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

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

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

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

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

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

ReparsePointTag (4 bytes): If **FILE_ATTRIBUTE_REPARSE_POINT** is set in the **FileAttributes** field, this field MUST contain a 32-bit unsigned integer value containing the reparse point tag that uniquely identifies the owner of the reparse point. section 2.1.2.1 contains more details on reparse tags.

FileId (8 bytes): The 64-bit file ID, as specified in section 2.1.9, for the file. For file systems that do not support a 64-bit file ID, this field MUST be set to 0, and MUST be ignored. For file systems which do not explicitly store directory entries named ".." (synonymous with the parent directory), an implementation MAY set this field to 0 for the entry named "..", and this value MUST be ignored. <124>

FileId128 (16 bytes): The 128-bit file ID, as specified in section 2.1.10, of the file. For file systems that do not support a 128-bit file ID, this field MUST be set to 0, and MUST be ignored.

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

This operation returns a status code as specified in section 2.2. Upon success, the status code returned by the function that processes this file information class is STATUS_SUCCESS. The most common error codes are listed in the following table.

Expand table

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

2.4.17 FileIdBothDirectoryInformation

Article12/14/2021

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

This information class returns a list that contains a **FILE_ID_BOTH_DIR_INFORMATION** data element for each file or directory within the target directory.

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

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

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										Reserved1												ShortName (24 bytes)									
...																															
...																															
...																						Reserved2									

FileId
...
FileName (variable)
...

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

FileIndex (4 bytes): A 32-bit unsigned integer that contains the byte offset of the file within the parent directory. For file systems in which the position of a file within the parent directory is not fixed and can be changed at any time to maintain sort order, this field SHOULD be set to 0 and MUST be ignored. <110>

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

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

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

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

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

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

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

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

EaSize (4 bytes): If **FILE_ATTRIBUTE_REPARSE_POINT** is set in the **FileAttributes** field, this field MUST contain a reparse tag as specified in section 2.1.2.1. Otherwise, this field is a 32-bit unsigned integer that contains the combined length, in bytes, of the extended attributes (EA) for the file.

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

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

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

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

FileId (8 bytes): The 64-bit file ID, as specified in section 2.1.9, for the file. For file systems that do not support a 64-bit file ID, this field MUST be set to 0, and MUST be ignored. For file systems which do not explicitly store directory entries named ".." (synonymous with the parent directory), an implementation MAY set this field to 0 for the entry named "..", and this value MUST be ignored. <111>

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

This operation returns a status code as specified in section 2.2. Upon success, the status code returned by the function that processes this file information class is STATUS_SUCCESS. The most common error codes are listed in the following table.

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

2.4.18 FileIdExtdDirectoryInformation

Article04/27/2022

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

This information class returns a list that contains a **FILE_ID_EXTD_DIR_INFORMATION** data element for each file or directory within the target directory.

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

A **FILE_ID_EXTD_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																															
ReparsePointTag																															
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_EXTD_DIR_INFORMATION entry is located, if multiple entries are present in the buffer. This member MUST be zero if no other entries follow this one. An implementation MUST use this value to determine the location of the next entry (if multiple entries are present in a buffer).

FileIndex (4 bytes): A 32-bit unsigned integer that contains the byte offset of the file within the parent directory. For file systems in which the position of a file within the parent directory is not fixed and can be changed at any time to maintain sort order, this field SHOULD be set to 0 and MUST be ignored. <112>

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

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

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

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

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

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

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

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

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

ReparsePointTag (4 bytes): If **FILE_ATTRIBUTE_REPARSE_POINT** is set in the **FileAttributes** field, this field MUST contain a 32-bit unsigned integer value containing the reparse point tag that uniquely identifies the owner of the reparse point. section 2.1.2.1 contains more details on reparse tags.

FileId (16 bytes): The 128-bit file ID, as specified in section 2.1.10, of the file. For file systems that do not support a 128-bit file ID, this field MUST be set to 0, and MUST be ignored.

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

This operation returns a status code as specified in section 2.2. Upon success, the status code returned by the function that processes this file information class is STATUS_SUCCESS. The most common error codes are listed in the following table.

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

2.4.24 FileIdFullDirectoryInformation

Article04/07/2025

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

This information class returns a list that contains a **FILE_ID_FULL_DIR_INFORMATION** data element for each file or directory within the target directory.

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

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

Expand table

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 field SHOULD<128> be zero if no other entries follow this one. An implementation MUST use this value to determine the location of the next entry (if multiple entries are present in a buffer).

FileIndex (4 bytes): A 32-bit unsigned integer that contains the byte offset of the file within the parent directory. For file systems in which the position of a file within the parent directory is not fixed and can be changed at any time to maintain sort order, this field SHOULD be set to 0 and MUST be ignored.<129>

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

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

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

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

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

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

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

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

EaSize (4 bytes): If **FILE_ATTRIBUTE_REPARSE_POINT** is set in the **FileAttributes** field, this field MUST contain a reparse tag as specified in section 2.1.2.1. Otherwise, this field is a 32-bit unsigned integer that contains the combined length, in bytes, of the extended attributes (EA) for the file.

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

FileId (8 bytes): The 64-bit file ID, as specified in section 2.1.9, for the file. For file systems that do not support a 64-bit file ID, this field MUST be set to 0, and MUST be ignored. For file systems which do not

explicitly store directory entries named ".." (synonymous with the parent directory), an implementation MAY set this field to 0 for the entry named "..", and this value MUST be ignored.<130>

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

This operation returns a status code as specified in section 2.2. Upon success, the status code returned by the function that processes this file information class is STATUS_SUCCESS. The most common error codes are listed in the following table.

Expand table

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

2.4.25 FileIdGlobalTxDirectoryInformation

Article04/07/2025

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

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

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

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

Expand table

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

FileNameLength
FileId
...
LockingTransactionId (16 bytes)
...
...
TxInfoFlags
FileName (variable)
...

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

FileIndex (4 bytes): A 32-bit unsigned integer that contains the byte offset of the file within the parent directory. For file systems in which the position of a file within the parent directory is not fixed and can be changed at any time to maintain sort order, this field SHOULD be set to 0 and MUST be ignored. <131>

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

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

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

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

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

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

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

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

FileId (8 bytes): The 64-bit file ID, as specified in section 2.1.9, for the file. For file systems that do not support a 64-bit file ID, this field MUST be set to 0, and MUST be ignored. For file systems which do not explicitly store directory entries named ".." (synonymous with the parent directory), an implementation MAY set this field to 0 for the entry named "..", and this value MUST be ignored.<132>

LockingTransactionId (16 bytes): A **GUID** value that is the ID of the transaction that has this file locked for modification. This number is generated and assigned by the file system. If the FILE_ID_GLOBAL_TX_DIR_INFO_FLAG_WRITELOCKED flag is not set in the **TxInfoFlags** field, this field MUST be ignored.

TxInfoFlags (4 bytes): A 32-bit unsigned integer that contains a bitmask of flags that indicate the transactional visibility of the file. The value of this field MUST be a bitwise OR of zero or more of the following values. Any flag values not explicitly mentioned here can be set to any value and MUST be ignored. If the FILE_ID_GLOBAL_TX_DIR_INFO_FLAG_WRITELOCKED flag is not set, the other flags MUST NOT be set. If flags other than FILE_ID_GLOBAL_TX_DIR_INFO_FLAG_WRITELOCKED are set, FILE_ID_GLOBAL_TX_DIR_INFO_FLAG_WRITELOCKED MUST be set.

Expand table

Value	Meaning
FILE_ID_GLOBAL_TX_DIR_INFO_FLAG_WRITELOCKED 0x00000001	The file is locked for modification by a transaction. The transaction's ID MUST be contained in the **LockingTransactionId** field if this flag is set.
FILE_ID_GLOBAL_TX_DIR_INFO_FLAG_VISIBLE_TO_TX 0x00000002	The file is visible to transacted enumerators of the directory whose transaction ID is in the **LockingTransactionId** field.
FILE_ID_GLOBAL_TX_DIR_INFO_FLAG_VISIBLE_OUTSIDE_TX 0x00000004	The file is visible to transacted enumerators of the directory other than the one whose transaction ID is in the **LockingTransactionId** field, and it is visible to non-transacted enumerators of the directory.

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

This operation returns a status code as specified in section 2.2. Upon success, the status code returned by the function that processes this file information class is STATUS_SUCCESS. The most common error codes are listed in the following table.

Expand table

Error code	Meaning
STATUS_NOT_SUPPORTED 0xC00000BB	The request is not supported.

2.4.26 FileIdInformation

06/10/2025

This information class is used to query the volume serial number and fileid information for a file.

A **FILE_ID_INFORMATION** data element, defined as follows, is provided by the server.

Expand table

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

VolumeSerialNumber (8 bytes): A 64-bit unsigned integer that contains the serial number of the volume where the file is located.

FileId (16 bytes): The 128-bit file ID, as specified in section 2.1.10, of the file. For file systems that do not support a 128-bit file ID, this field MUST be set to 0, and MUST be ignored.

This operation returns a status code as specified in section 2.2. Upon success, the status code returned by the function that processes this file information class is STATUS_SUCCESS. The most common error codes are listed in the following table.

Expand table

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

2.4.27 FileInternalInformation

Article04/07/2025

This information class is used to query for the file system's 64-bit file ID, as specified in section 2.1.9.

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

Expand table

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
IndexNumber																															
...																															

IndexNumber (8 bytes): The 64-bit file ID for the file. For file systems that do not support a 64-bit file ID, this field MUST be set to 0, and MUST be ignored. <133>

This operation returns a status code as specified in section 2.2. Upon success, the status code returned by the function that processes this file information class is STATUS_SUCCESS. The most common error codes are listed in the following table.

Expand table

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

2.4.28 FileLinkInformation

Article04/07/2025

This information class is used to create a hard link to an existing file.<134> The Server Message Block (SMB) Protocol [MS-SMB] and the Server Message Block (SMB) Version 2 Protocol [MS-SMB2] implement unique structure variants:

- **FILE_LINK_INFORMATION_TYPE_1**, as specified in section 2.4.28.1.
- **FILE_LINK_INFORMATION_TYPE_2**, as specified in section 2.4.28.2.

This operation returns a status code as specified in section 2.2. Upon success, the status code returned by the function that processes this file information class is STATUS_SUCCESS. The most common error codes are listed in the following table.

Expand table

Error code	Meaning
STATUS_INVALID_PARAMETER 0xC000000D	An invalid parameter was specified for the **RootDirectory** field.
STATUS_FILE_IS_A_DIRECTORY 0xC00000BA	The file that was specified is a directory.
STATUS_ACCESS_DENIED 0xC0000022	The object has been deleted.
STATUS_OBJECT_NAME_INVALID 0xC0000033	The object name is invalid for the target file system.
STATUS_TOO_MANY_LINKS 0xC0000265	An attempt was made to create more links on a file than the file system supports.
STATUS_OBJECT_NAME_COLLISION 0xC0000035	The specified name already exists and **ReplaceIfExists** is zero.
STATUS_INFO_LENGTH_MISMATCH 0xC0000004	The specified information record length does not match the length that is required for the specified information class.
STATUS_NOT_SUPPORTED 0xC00000BB	The request is not supported.

2.4.28.1 FileLinkInformation for the SMB Protocol

Article04/07/2025

This information class is used to create a hard link to an existing file via the SMB Protocol as specified in [MS-SMB].

A **FILE_LINK_INFORMATION_TYPE_1** data element, defined as follows, is provided by the client.

Expand table

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
ReplaceIfExists								Reserved																							
RootDirectory																															
FileNameLength																															
FileName (variable)																															
...																															

ReplaceIfExists (1 byte): A Boolean (section 2.1.8) value. Set to TRUE to indicate that if the link already exists, it SHOULD be replaced with the new link. Set to FALSE to indicate that the link creation operation MUST fail if the link already exists.

Reserved (3 bytes): This field SHOULD be set to zero by the client and MUST be ignored by the server.

RootDirectory (4 bytes): A 32-bit unsigned integer that contains the file handle for the directory where the link is to be created. For network operations, this value MUST always be zero.

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

FileName (variable): A sequence of Unicode characters that contains the name to be assigned to the newly created link. When working with the **FileName** field, the **FileNameLength** field is used to determine the length of the file name rather than assuming the presence of a trailing null delimiter. If the **RootDirectory** field is zero, this field MUST specify a full pathname to the link to be created. For network operations, this pathname is relative to the root of the share. If the **RootDirectory** field is not zero, this field MUST specify a pathname, relative to **RootDirectory**, for the link name.

2.4.28.2 FileLinkInformation for the SMB2 Protocol

Article04/07/2025

This information class is used to create a hard link to an existing file via the SMB Version 2 Protocol, as specified in [MS-SMB2].

A **FILE_LINK_INFORMATION_TYPE_2** data element, defined as follows, is provided by the client.

Expand table

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
ReplaceIfExists								Reserved																							
...																															
RootDirectory																															
...																															
FileNameLength																															
FileName (variable)																															
...																															

ReplaceIfExists (1 byte): A Boolean (section 2.1.8) value. Set to TRUE to indicate that if the link already exists, it SHOULD be replaced with the new link. Set to FALSE to indicate that the link creation operation MUST fail if the link already exists.

Reserved (7 bytes): Reserved for alignment. This field can contain any value and MUST be ignored.

RootDirectory (8 bytes): A 64-bit unsigned integer that contains the file handle for the directory where the link is to be created. For network operations, this value MUST be zero.

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

FileName (variable): A sequence of Unicode characters containing the name to be assigned to the newly created link. When working with this field, the **FileNameLength** field is used to determine the length of the file name rather than assuming the presence of a trailing null delimiter. If the **RootDirectory** field is zero, this field MUST specify a full pathname to the link to be created. For network operations, this pathname is relative to the root of the share. If the **RootDirectory** field is not zero, this field MUST specify a pathname, relative to **RootDirectory**, for the link name.

2.4.29 FileMailslotQueryInformation

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

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

Expand table

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
MaximumMessageSize																															
MailslotQuota																															
NextMessageSize																															
MessagesAvailable																															
ReadTimeout																															
...																															

MaximumMessageSize (4 bytes): A 32-bit unsigned integer that contains the maximum size of a single message that can be written to the mailslot, in bytes. To specify that the message can be of any size, set this value to zero.

MailslotQuota (4 bytes): A 32-bit unsigned integer that contains the quota, in bytes, for the mailslot. The mailslot quota specifies the in-memory pool quota that is reserved for writes to this mailslot.

NextMessageSize (4 bytes): A 32-bit unsigned integer that contains the next message size, in bytes.

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

ReadTimeout (8 bytes): A 64-bit signed integer that contains the time a read operation can wait for a message to be written to the mailslot before a time-out occurs in milliseconds. The value of this field MUST be (-1) or greater than or equal to 0. A value of (-1) requests that the read wait forever for a message, without timing out. A value of 0 requests that the read not wait and return immediately whether a pending message is available to be read or not.

This operation returns a status code as specified in section 2.2. Upon success, the status code returned by the function that processes this file information class is STATUS_SUCCESS. The most common error codes are listed in the following table.

Expand table

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

Last updated on 11/21/2025

2.4.25 FileMailslotSetInformation

Article04/27/2022

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

A **FILE_MAILSLOT_SET_INFORMATION** data element, defined as follows, is provided by the caller.

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
ReadTimeout																															
...																															

ReadTimeout (8 bytes): A 64-bit signed integer that contains the time that a read operation can wait for a message to be written to the mailslot before a time-out occurs as follows:

- A positive value specifies the operation time-out as an absolute system time on the server, represented as a count of 100-nanosecond intervals since January 1, 1601.
- A negative value specifies the number of 100-nanosecond intervals for the operation to time out relative to the current server time.
- A value of -1 (0xFFFFFFFFFFFFFFFF) requests that the read wait forever for a message without timing out.
- A value of zero sends a request that the read not wait and return immediately, whether a pending message is available to be read or not.

This operation returns a status code as specified in section 2.2. Upon success, the status code returned by the function that processes this file information class is STATUS_SUCCESS. The most common error codes are listed in the following table.

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

2.4.31 FileModeInformation

Article04/07/2025

The **FileModeInformation** information class is used to query or set the mode of the file. The mode returned by a query corresponds to the **CreateOptions** used in the initial create operation, modified by any set FileModeInformation operations performed since the create operation.<135>

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

Expand table

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

Mode (4 bytes): A 32-bit unsigned integer that specifies how the file will subsequently be accessed.

Expand table

Value	Meaning
FILE_WRITE_THROUGH 0x00000002	When set, any system services, file system drivers (FSDs), and drivers that write data to the file are required to actually transfer the data into the file before any requested write operation is considered complete.
FILE_SEQUENTIAL_ONLY 0x00000004	This is a hint that informs the cache that it SHOULD<136> optimize for sequential access. Non-sequential access of the file can result in performance degradation.
FILE_NO_INTERMEDIATE_BUFFERING 0x00000008	When set, the file cannot be cached or buffered in a driver's internal buffers.
FILE_SYNCHRONOUS_IO_ALERT 0x00000010	When set, all operations on the file are performed synchronously. Any wait on behalf of the caller is subject to premature termination from alerts. This flag also causes the I/O system to maintain the file position context.
FILE_SYNCHRONOUS_IO_NONALERT 0x00000020	When set, all operations on the file are performed synchronously. Wait requests in the system to synchronize I/O queuing and completion are not subject to alerts. This flag also causes the I/O system to maintain the file position context.
FILE_DELETE_ON_CLOSE 0x00001000	This flag is not implemented and is always returned as not set.

This operation returns a status code as specified in section 2.2. Upon success, the status code returned by the function that processes this file information class is STATUS_SUCCESS. The most common error codes are listed in the following table.

Expand table

Error code	Meaning
STATUS_INFO_LENGTH_MISMATCH 0xC0000004	The specified information record length does not match the length that is required for the specified information class.
STATUS_INVALID_PARAMETER	An attempt to set the file mode returns STATUS_INVALID_PARAMETER in any of the following cases: • The **Mode** field contains any flag other than FILE_WRITE_THROUGH, FILE_SEQUENTIAL_ONLY, FILE_SYNCHRONOUS_IO_ALERT, or FILE_SYNCHRONOUS_IO_NONALERT. • FILE_SYNCHRONOUS_IO_ALERT or FILE_SYNCHRONOUS_IO_NONALERT is set and the file was not opened for synchronous I/O. • Neither FILE_SYNCHRONOUS_IO_ALERT nor FILE_SYNCHRONOUS_IO_NONALERT are set and the file was opened for synchronous I/O. • FILE_SYNCHRONOUS_IO_ALERT and FILE_SYNCHRONOUS_IO_NONALERT are both set.

2.4.32 FileNameInformation

Article04/07/2025

This information class is used locally to query the name of a file. This information class returns a **FILE_NAME_INFORMATION** data element containing an absolute pathname (section 2.1.5).

This operation returns a status code as specified in section 2.2. Upon success, the status code returned by the function that processes this file information class is STATUS_SUCCESS. The most common error codes are listed in the following table.

Expand table

Error code	Meaning
STATUS_NOT_SUPPORTED 0xC00000BB	The resource is not supported.
STATUS_BUFFER_OVERFLOW 0x80000005	The output buffer was filled before the complete name could be returned.

2.4.28 FileNamesInformation

Article05/28/2021

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

This information class returns a list that contains a **FILE_NAMES_INFORMATION** data element for each file or directory within the target directory.

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

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

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																															
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_NAMES_INFORMATION entry is located, if multiple entries are present in a buffer. This member MUST be zero if no other entries follow this one. An implementation MUST use this value to determine the location of the next entry (if multiple entries are present in a buffer).

FileIndex (4 bytes): A 32-bit unsigned integer that contains the byte offset of the file within the parent directory. For file systems in which the position of a file within the parent directory is not fixed and can be changed at any time to maintain sort order, this field SHOULD be set to 0 and MUST be ignored. <122>

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

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

This operation returns a status code as specified in section 2.2. Upon success, the status code returned by the function that processes this file information class is STATUS_SUCCESS. The most common error codes are listed in the following table.

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

2.4.34 FileNetworkOpenInformation

Article04/07/2025

This information class is used to query for information that is commonly needed when a file is opened across a network.<138>

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

Expand table

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
CreationTime																															
...																															
LastAccessTime																															
...																															
LastWriteTime																															
...																															
ChangeTime																															
...																															
AllocationSize																															
...																															
EndOfFile																															
...																															
FileAttributes																															
Reserved																															

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

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

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

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

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

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

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

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

This operation returns a status code as specified in section 2.2. Upon success, the status code returned by the function that processes this file information class is STATUS_SUCCESS. The most common error codes are listed in the following table.

Expand table

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

2.4.35 FileNormalizedNameInformation

Article04/07/2025

This information class is used to query the normalized name of a file. A normalized name is an absolute pathname where each short name component has been replaced with the corresponding long name component, and each name component uses the exact letter casing stored on disk. This information class returns a FILE_NAME_INFORMATION data element containing an absolute pathname, as specified in section 2.1.7. <139>

This operation returns a status code as specified in section 2.2. Upon success, the status code returned by the function that processes this file information class is STATUS_SUCCESS. The most common error codes are listed in the following table.

Expand table

Error Code	Meaning
STATUS_NOT_SUPPORTED 0xC00000BB	The resource is not supported.
STATUS_BUFFER_OVERFLOW 0x80000005	The output buffer was filled before the complete name could be returned.

2.4.36 FileObjectIdInformation

Article04/07/2025

This information class is used locally to query object ID information for the files in a directory on a volume. The query MUST fail if the file system does not support object IDs.<140>

The data returned to the caller will take one of two forms. The choice of which data structure to use, and the interpretation of the data within it, is application-specific. An application implementer chooses one of the following two data elements as the structure for its object ID information data.<141>

- FILE_OBJECTID_INFORMATION_TYPE_1 (section 2.4.36.1).
- FILE_OBJECTID_INFORMATION_TYPE_2 (section 2.4.36.2).

This operation returns a status code as specified in section 2.2. Upon success, the status code returned by the function that processes this file information class is STATUS_SUCCESS. The most common error codes are listed in the following table.

Expand table

Error code	Meaning
STATUS_INVALID_DEVICE_REQUEST 0xC0000010	The target file system does not implement this functionality.
STATUS_INVALID_INFO_CLASS 0xC0000003	The specified information class is not a valid information class for the specified object.
STATUS_INFO_LENGTH_MISMATCH 0xC0000004	The specified information record length does not match the length that is required for the specified information class.
STATUS_INVALID_PARAMETER 0xC000000D	The file specified is not a valid parameter.
STATUS_NO_SUCH_FILE 0xC000000F	The file does not exist.
STATUS_NO_MORE_FILES 0x80000006	No more files were found which match the file specification.

Error code	Meaning
STATUS_BUFFER_OVERFLOW 0x80000005	The output buffer was filled before all of the **ObjectID** information could be returned. Only complete **FILE_OBJECTID_INFORMATION** structures are returned.

2.4.36.1 FILE_OBJECTID_INFORMATION_TYPE_1

Article04/07/2025

A **FILE_OBJECTID_INFORMATION_TYPE_1** data element is as follows.

Expand table

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 (16 bytes)																															
...																															
...																															
BirthVolumeId (16 bytes)																															
...																															
...																															
BirthObjectId (16 bytes)																															
...																															
...																															
DomainId (16 bytes)																															
...																															
...																															

FileReferenceNumber (8 bytes): The 64-bit file ID, as specified in section 2.1.9, for the file. For file systems that do not support a 64-bit file ID, this field MUST be set to 0, and MUST be ignored.

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 might 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 might not be the same as the **ObjectId** member at present.<142>

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

2.4.36.2 FILE_OBJECTID_INFORMATION_TYPE_2

Article04/07/2025

A **FILE_OBJECTID_INFORMATION_TYPE_2** data element is as follows.

Expand table

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 (16 bytes)																															
...																															
...																															
ExtendedInfo (48 bytes)																															
...																															
...																															

FileReferenceNumber (8 bytes): The 64-bit file ID, as specified in section 2.1.9, for the file. For file systems that do not support a 64-bit file ID, this field MUST be set to 0, and MUST be ignored.

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.

2.4.32 FilePipeInformation

Article04/27/2022

This information class is used to query or set information on a named pipe that is not specific to one end of the pipe or another.

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

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
ReadMode																															
CompletionMode																															

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

Value	Meaning
FILE_PIPE_BYTE_STREAM_MODE 0x00000000	If this value is specified, data MUST be read from the pipe as a stream of bytes.
FILE_PIPE_MESSAGE_MODE 0x00000001	If this value is specified, data MUST be read from the pipe as a stream of messages.

If this field is set to FILE_PIPE_BYTE_STREAM_MODE, any attempt to subsequently change it MUST fail with a STATUS_INVALID_PARAMETER error code.

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

Value	Meaning
FILE_PIPE_QUEUE_OPERATION 0x00000000	If this value is specified, blocking mode MUST be enabled. When the pipe is being connected, read to, or written from, the operation is not completed until there is data to read, all data is written, or a client is connected. Use of this mode can result in the server waiting indefinitely for a client process to perform an action.
FILE_PIPE_COMPLETE_OPERATION 0x00000001	If this value is specified, non-blocking mode MUST be enabled. When the pipe is being connected, read to, or written from, the operation completes immediately.

This operation returns a status code as specified in section 2.2. Upon success, the status code returned by the function that processes this file information class is STATUS_SUCCESS. The most common error codes are listed in the following table.

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

Error code	Meaning
STATUS_INVALID_PARAMETER 0xC000000D	An invalid parameter was passed to a service or function. When setting the FilePipeInformation information level, STATUS_INVALID_PARAMETER will be returned: • If the **ReadMode** field is set to FILE_PIPE_BYTE_STREAM_MODE and a subsequent set operation attempts to set the **ReadMode** field to any value other than FILE_PIPE_BYTE_STREAM_MODE. • If the value of the **ReadMode** field is not equal to FILE_PIPE_MESSAGE_MODE or FILE_PIPE_BYTE_STREAM_MODE. • If the value of the **CompletionMode** field is not equal to FILE_PIPE_QUEUE_OPERATION or FILE_PIPE_COMPLETE_OPERATION.

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

2.4.33 FilePipeLocalInformation

Article06/24/2021

This information class is used to query information on a named pipe that is associated with the end of the pipe that is being queried.

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

0	1	2	3	4	5	6	7	8	9	10	1	2	3	4	5	6	7	8	9	20	1	2	3	4	5	6	7	8	9	30	1
NamedPipeType																															
NamedPipeConfiguration																															
MaximumInstances																															
CurrentInstances																															
InboundQuota																															
ReadDataAvailable																															
OutboundQuota																															
WriteQuotaAvailable																															
NamedPipeState																															
NamedPipeEnd																															

NamedPipeType (4 bytes): A 32-bit unsigned integer that contains the named pipe type. MUST be one of the following.

Value	Meaning
FILE_PIPE_BYTE_STREAM_TYPE 0x00000000	If this value is specified, data MUST be read from the pipe as a stream of bytes.
FILE_PIPE_MESSAGE_TYPE 0x00000001	If this flag is specified, data MUST be read from the pipe as a stream of messages.

NamedPipeConfiguration (4 bytes): A 32-bit unsigned integer that contains the named pipe configuration. MUST be one of the following.

Value	Meaning
FILE_PIPE_INBOUND 0x00000000	If this value is specified, the flow of data in the pipe goes from client to server only.
FILE_PIPE_OUTBOUND 0x00000001	If this value is specified, the flow of data in the pipe goes from server to client only.

Value	Meaning
FILE_PIPE_FULL_DUPLEX 0x00000002	If this value is specified, the pipe is bi-directional; both server and client processes can read from and write to the pipe.

MaximumInstances (4 bytes): A 32-bit unsigned integer that contains the maximum number of instances that can be created for this pipe.

CurrentInstances (4 bytes): A 32-bit unsigned integer that contains the number of current named pipe instances.

InboundQuota (4 bytes): A 32-bit unsigned integer that contains the inbound quota, in bytes, for the named pipe. The inbound quota is the size of the buffer reserved for inbound transfer of data on the pipe.

ReadDataAvailable (4 bytes): A 32-bit unsigned integer that contains the bytes of data available to be read from the named pipe.

OutboundQuota (4 bytes): A 32-bit unsigned integer that contains the outbound quota, in bytes, for the named pipe. The outbound quota is the size of the buffer reserved for outbound transfer of data on the pipe.

WriteQuotaAvailable (4 bytes): A 32-bit unsigned integer that contains the write quota, in bytes, for the named pipe. If the **NamedPipeEnd** field is set to FILE_PIPE_CLIENT_END, the **WriteQuotaAvailable** field is the remaining **InboundQuota** field available. If the **NamedPipeEnd** field is set to FILE_PIPE_SERVER_END, the **WriteQuotaAvailable** field is the remaining **OutboundQuota** field available.

NamedPipeState (4 bytes): A 32-bit unsigned integer that contains the named pipe state that specifies the connection status for the named pipe. MUST be one of the following.

Value	Meaning
FILE_PIPE_DISCONNECTED_STATE 0x00000001	Named pipe is disconnected.
FILE_PIPE_LISTENING_STATE 0x00000002	Named pipe is waiting to establish a connection.
FILE_PIPE_CONNECTED_STATE 0x00000003	Named pipe is connected.
FILE_PIPE_CLOSING_STATE 0x00000004	Named pipe is in the process of being closed.

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

Value	Meaning
FILE_PIPE_CLIENT_END 0x00000000	This is the client end of a named pipe.

Value	Meaning
FILE_PIPE_SERVER_END 0x00000001	This is the server end of a named pipe.

This operation returns a status code as specified in section 2.2. Upon success, the status code returned by the function that processes this file information class is STATUS_SUCCESS. The most common error codes are listed in the following table.

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

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

2.4.34 FilePipeRemoteInformation

Article05/28/2021

This information class is used to query information on a named pipe that is associated with the client end of the pipe that is being queried. Remote information is not available for local pipes or for the server end of a remote pipe. Therefore, this information class is usable only by the client to retrieve information associated with its end of the pipe.

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

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

CollectDataTime (8 bytes): A 64-bit signed integer that MUST contain the maximum amount of time counted in 100-nanosecond intervals that will elapse before transmission of data from the client machine to the server.

MaximumCollectionCount (4 bytes): A 32-bit unsigned integer that MUST contain the maximum size, in bytes, of data that will be collected on the client machine before transmission to the server.

This operation returns a status code as specified in section 2.2. Upon success, the status code returned by the function that processes this file information class is STATUS_SUCCESS. The most common error codes are listed in the following table.

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

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

2.4.40 FilePositionInformation

Article04/07/2025

This information class is used to query or set the position of the file pointer within a file.<143>

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

Expand table

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
CurrentByteOffset																															
...																															

CurrentByteOffset (8 bytes): A 64-bit signed integer that MUST contain the offset, in bytes, of the file pointer from the beginning of the file. A unique offset value is maintained for each open of a file. When setting the position, only values greater than or equal to zero are valid. If the given file was opened using the FILE_NO_INTERMEDIATE_BUFFERING flag, the offset that is being set SHOULD be aligned to a sector boundary. This value SHOULD<144> be updated by read and write operations if the given file was opened using the FILE_SYNCHRONOUS_IO_ALERT or FILE_SYNCHRONOUS_IO_NONALERT flags.

This operation returns a status code as specified in section 2.2. Upon success, the status code returned by the function that processes this file information class is STATUS_SUCCESS. The most common error codes are listed in the following table.

Expand table

Error code	Meaning
STATUS_INFO_LENGTH_MISMATCH 0xC0000004	The specified information record length does not match the length that is required for the specified information class.
STATUS_INVALID_PARAMETER 0xC000000D	Returned when setting the offset if the **CurrentByteOffset** is negative or the file was opened using the FILE_NO_INTERMEDIATE_BUFFERING flag and **CurrentByteOffset** is not aligned to a sector boundary.

2.4.41 FileQuotaInformation

This information class is used to query or to set file quota information for a volume. For queries, an optional buffer of FILE_GET_QUOTA_INFORMATION (section 2.4.41.1) data elements is provided by the client to specify the SIDs for which quota information is requested. If the **FILE_GET_QUOTA_INFORMATION** buffer is not specified, information for all quotas is returned. A buffer of **FILE_QUOTA_INFORMATION** data elements is returned by the server. For sets, **FILE_QUOTA_INFORMATION** data elements are populated and sent by the client, as specified in [MS-SMB] section 2.2.7.6.1 and [MS-SMB2] section 3.2.4.15.<145>

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

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

Expand table

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																															
SidLength																															
ChangeTime																															
...																															
QuotaUsed																															
...																															
QuotaThreshold																															
...																															
QuotaLimit																															
...																															
Sid (variable)																															
...																															

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

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

ChangeTime (8 bytes): The last time that the quota was changed; see section 2.1.1. This value MUST be greater than or equal to 0x0000000000000000. When setting quota information, the server MUST ignore the value of this field.

QuotaUsed (8 bytes): A 64-bit signed integer that contains the amount of quota used by this user, in bytes. This value MUST be greater than or equal to 0x0000000000000000. When setting quota information, the server MUST ignore the value of this field.

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

QuotaLimit (8 bytes): A 64-bit signed integer that contains the disk quota limit, in bytes, on this volume for this user. This field MUST be set to a 64-bit integer value greater than or equal to zero to set a disk quota limit for this user on this volume, to -1 to specify that no quota limit is set for this user, or to -2 to delete the quota entry for the user.

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

This operation returns a status code as specified in section 2.2. Upon success, the status code returned by the function that processes this file information class is STATUS_SUCCESS. The most common error codes are listed in the following table.

Expand table

Error code	Meaning
STATUS_INVALID_DEVICE_REQUEST 0xC0000010	The target file system does not implement this functionality.
STATUS_INVALID_INFO_CLASS 0xC0000003	The specified information class is not a valid information class for the specified object.
STATUS_INFO_LENGTH_MISMATCH 0xC0000004	The specified information record length does not match the length that is required for the specified information class.
STATUS_INVALID_PARAMETER 0xC000000D	The SID or SID Length specified is not a valid parameter.
STATUS_NO_SUCH_FILE 0xC000000F	For query operations, indicates that no **FILE_QUOTA_INFORMATION** data elements were returned that matched the input criteria.
STATUS_BUFFER_TOO_SMALL 0xC0000023	The buffer is too small to contain the entry. No information has been written to the buffer.

Last updated on 11/21/2025

2.4.41.1 FILE_GET_QUOTA_INFORMATION

Article04/07/2025

This structure is used to provide the list of SIDs for which quota query information is requested.

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

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

Expand table

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																															
SidLength																															
Sid (variable)																															
...																															

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

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

Sid (variable): SID for this user. SIDs are sent in little-endian format and require no padding. The format of a SID is as specified in [MS-DTYP] section 2.4.2.2.

2.4.42 FileRenameInformation

Article04/07/2025

This information class is used to rename a file. The data element provided by the client takes one of two forms, depending on whether it is embedded within SMB or SMB2. The structure definitions are as follows:

- FILE_RENAME_INFORMATION_TYPE_1 for the SMB protocol (section 2.4.42.1).
- FILE_RENAME_INFORMATION_TYPE_2 for the SMB2 protocol (section 2.4.42.2).

This operation returns a status code as specified in section 2.2. Upon success, the status code returned by the function that processes this file information class is STATUS_SUCCESS. The most common error codes are listed in the following table.

Expand table

Error code	Meaning
STATUS_INVALID_PARAMETER 0xC000000D	An invalid parameter was passed for **FileName** or **FileNameLength**, or the **RootDirectory** field value was nonzero for a network operation.
STATUS_ACCESS_DENIED 0xC0000022	The handle was not opened with delete access, or the target file was open and **ReplaceIfExists** is nonzero.
STATUS_NOT_SAME_DEVICE 0xC00000D4	The destination file of a rename request is located on a different device than the source of the rename request.
STATUS_OBJECT_NAME_INVALID 0xC0000033	The object name is invalid for the target file system.
STATUS_OBJECT_NAME_COLLISION 0xC0000035	The specified name already exists and **ReplaceIfExists** is zero.
STATUS_INFO_LENGTH_MISMATCH 0xC0000004	The specified information record length does not match the length that is required for the specified information class.

2.4.42.1 FileRenameInformation for SMB

Article04/07/2025

This information class is used to rename a file from within the SMB Protocol, as specified in [MS-SMB].

A **FILE_RENAME_INFORMATION_TYPE_1** data element, defined as follows, is provided by the client.

Expand table

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
ReplaceIfExists								Reserved																							
RootDirectory																															
FileNameLength																															
FileName (variable)																															
...																															

ReplaceIfExists (1 byte): A Boolean (section 2.1.8) value. Set to TRUE to indicate that if a file with the given name already exists, it SHOULD be replaced with the given file. Set to FALSE to indicate that the rename operation MUST fail if a file with the given name already exists.

Reserved (3 bytes): Reserved area for alignment. This field can contain any value and MUST be ignored.

RootDirectory (4 bytes): A 32-bit unsigned integer that contains the file handle for the directory to which the new name of the file is relative. For network operations, this value MUST be zero.

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

FileName (variable): A sequence of Unicode characters containing the new file name of type **Filename** (section 2.1.5.2). When working with this field, use **FileNameLength** to determine the length of the file name rather than assuming the presence of a trailing null delimiter.

2.4.42.2 FileRenameInformation for SMB2

Article04/07/2025

This information class is used to rename a file from within the SMB2 Protocol [MS-SMB2].

A **FILE_RENAME_INFORMATION_TYPE_2** data element, defined as follows, is provided by the client.

Expand table

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
ReplaceIfExists								Reserved																							
...																															
RootDirectory																															
...																															
FileNameLength																															
FileName (variable)																															
...																															
Padding (variable)																															

ReplaceIfExists (1 byte): A Boolean (section 2.1.8) value. Set to TRUE to indicate that if a file with the given name already exists, it SHOULD be replaced with the given file. Set to FALSE to indicate that the rename operation MUST fail if a file with the given name already exists.

Reserved (7 bytes): Reserved area for alignment. This field can contain any value and MUST be ignored.

RootDirectory (8 bytes): A 64-bit unsigned integer that contains the file handle for the directory to which the new name of the file is relative. For network operations, this value MUST always be zero.

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

FileName (variable): A sequence of Unicode characters containing the new name of the file. When working with this field, use **FileNameLength** to determine the length of the file name rather than assuming the presence of a trailing null delimiter.

Padding (variable): Length of this field MUST be the number of bytes required to make the size of this structure at least 24. This field MAY be set to 0 and MUST be ignored on receipt.

2.4.43 FileRenameInformationEx

Article04/07/2025

This information class is used to rename a file.

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

Expand table

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
Flags																															
Reserved																															
RootDirectory																															
...																															
FileNameLength																															
FileName (variable)																															
...																															
Padding (variable)																															
...																															

Flags (4 bytes): A 32-bit field that specifies options on how the file is renamed.

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

Expand table

Value	Meaning
FILE_RENAME_REPLACE_IF_EXISTS 0x00000001	If set, indicates that if a file with the given name already exists, it SHOULD be replaced with the given file. If not set, indicates that the rename operation MUST fail if a file with the given name already exists.
FILE_RENAME_POSIX_SEMANTICS 0x00000002	If set and FILE_RENAME_REPLACE_IF_EXISTS is set, indicates that if a file with the given name already exists the file SHOULD be deleted using POSIX-style semantics. Existing handles to the replaced file continue to be valid. Any subsequent opens of the target name will open the renamed file, not the replaced file.
FILE_RENAME_SUPPRESS_PIN_STATE_INHERITANCE 0x00000004	If set, when renaming a file to a new directory, suppress any inheritance rules related to the FILE_ATTRIBUTE_PINNED and FILE_ATTRIBUTE_UNPINNED attributes.<146>

Value	Meaning
FILE_RENAME_SUPPRESS_STORAGE_RESERVE_INHERITANCE 0x00000008	If set, when renaming a file to a new directory, it suppresses any inheritance rules related to the storage reserve ID property of the file.<147>
FILE_RENAME_NO_INCREASE_AVAILABLE_SPACE 0x00000010	If set and FILE_RENAME_SUPPRESS_STORAGE_RESERVE_INHERITANCE is not set; when renaming a file to a new directory, automatically resize affected storage reserve areas as needed to prevent the user visible free space on the volume from increasing. Requires manage volume access.<148>
FILE_RENAME_NO_DECREASE_AVAILABLE_SPACE 0x00000020	if set and FILE_RENAME_SUPPRESS_STORAGE_RESERVE_INHERITANCE is not set; when renaming a file to a new directory, automatically resize affected storage reserve areas as needed to prevent the user visible free space on the volume from decreasing. Requires manage volume access.<149>
FILE_RENAME_PRESERVE_AVAILABLE_SPACE 0x00000030	Equivalent to specifying both FILE_RENAME_NO_INCREASE_AVAILABLE_SPACE and FILE_RENAME_NO_DECREASE_AVAILABLE_SPACE.<150>
FILE_RENAME_IGNORE_READONLY_ATTRIBUTE 0x00000040	If set and FILE_RENAME_REPLACE_IF_EXISTS is set; allow replacing a file even if the read-only attribute is set on the file.<151>
FILE_RENAME_FORCE_RESIZE_TARGET_SR 0x00000080	If set and FILE_RENAME_SUPPRESS_STORAGE_RESERVE_INHERITANCE is not set; when renaming a file to a new directory that is part of a different storage reserve area, always grow the target directory's storage reserve area by the full size of the file being renamed. Requires manage volume access.<152>
FILE_RENAME_FORCE_RESIZE_SOURCE_SR 0x00000100	If set and FILE_RENAME_SUPPRESS_STORAGE_RESERVE_INHERITANCE is not set; when renaming a file to a new directory that is part of a different storage reserve area, always shrink the source directory's storage reserve area by the full size of the file being renamed. Requires manage volume access.<153>
FILE_RENAME_FORCE_RESIZE_SR 0x00000180	Equivalent to specifying both FILE_RENAME_FORCE_RESIZE_TARGET_SR and FILE_RENAME_FORCE_RESIZE_SOURCE_SR.<154>

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

RootDirectory (8 bytes): A 64-bit unsigned integer that contains the file handle for the directory to which the new name of the file is relative. For network operations, this value MUST always be zero.

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

FileName (variable): A sequence of Unicode characters containing the new name of the file. When working with this field, use **FileNameLength** to determine the length of the file name rather than assuming the presence of a trailing null delimiter.

Padding (variable): Length of this field MUST be the number of bytes required to make the size of this structure at least 24. This field MAY be set to 0 and MUST be ignored on receipt.

2.4.44 FileReparsePointInformation

Article04/07/2025

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

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

Expand table

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): The 64-bit file ID, as specified in section 2.1.9, for the file.

Tag (4 bytes): A 32-bit unsigned integer value containing the reparse point tag that uniquely identifies the owner of the reparse point. Section 2.1.2.1 contains more details on reparse tags.

This operation returns a status code as specified in section 2.2. Upon success, the status code returned by the function that processes this file information class is STATUS_SUCCESS. The most common error codes are listed in the following table.

Expand table

Error code	Meaning
STATUS_INFO_LENGTH_MISMATCH 0xC0000004	The specified information record length does not match the length that is required for the specified information class.
STATUS_INVALID_DEVICE_REQUEST 0xC0000010	The target file system does not implement this functionality.
STATUS_INVALID_INFO_CLASS 0xC0000003	The specified information class is not a valid information class for the specified object.
STATUS_NO_SUCH_FILE 0xC000000F	No reparse points exist for the given file.
STATUS_BUFFER_OVERFLOW 0x80000005	The output buffer was filled before all of the FILE_REPARSE_POINT_INFORMATION structures could be returned; a partial structure might be returned.

2.4.45 FileSfioReserveInformation

Article04/07/2025

This information class is used locally to query or set reserved bandwidth for a file handle. Conceptually reserving bandwidth is effectively specifying the bytes per second to allocate to file IO.

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

Expand table

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
RequestsPerPeriod																															
Period																															
RetryFailures								Discardable								Reserved															
RequestSize																															
NumOutstandingRequests																															

RequestsPerPeriod (4 bytes): A 32-bit unsigned integer indicating the number of I/O requests that complete per period of time, as specified in the **Period** field. When setting bandwidth reservation, a value of 0 indicates to the file system that it MUST free any existing reserved bandwidth.

Period (4 bytes): A 32-bit unsigned integer that contains the period for reservation, which is the time from which I/O is issued to the kernel until the time the I/O is completed, specified in milliseconds.

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

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

Reserved (2 bytes): Reserved for alignment. This field can contain any value and MUST be ignored.

RequestSize (4 bytes): A 32-bit unsigned integer that indicates the minimum size of any individual I/O request that can be issued by an application using bandwidth reservation. When setting reservations, this field MUST be ignored by servers and SHOULD be set to 0 by clients.

NumOutstandingRequests (4 bytes): A 32-bit unsigned integer that indicates the number of RequestSize I/O requests allowed to be outstanding at any time. When setting reservations, this field MUST be ignored by servers and SHOULD be set to 0 by clients.

This operation returns a status code as specified in section 2.2. Upon success, the status code returned by the function that processes this file information class is STATUS_SUCCESS. The most common error codes are listed in the following table.

Expand table

Error code	Meaning
STATUS_NOT_SUPPORTED 0xC00000BB	The request is not supported.
STATUS_INFO_LENGTH_MISMATCH 0xC0000004	The specified information record length does not match the length that is required for the specified information class.

2.4.46 FileShortNameInformation

Article04/07/2025

This information class is used to change a file's short name. If the supplied name is of zero length, the file's existing short name, if any, SHOULD<155> be deleted. Otherwise, the supplied name MUST be a valid short name as specified in section 2.1.5.2.1 and be unique among all file names and short names in the same directory as the file being operated on. A caller changing the file's short name MUST have SeRestorePrivilege, as specified in [MS-LSAD] section 3.1.1.2.1.

A FILE_NAME_INFORMATION (section 2.1.7) data element containing an 8.3 file name (section 2.1.5.2.1) is provided by the client.

This operation returns a status code as specified in section 2.2. Upon success, the status code returned by the function that processes this file information class is STATUS_SUCCESS. The most common error codes are listed in the following table.

Expand table

Error code	Meaning
STATUS_MEDIA_WRITE_PROTECTED 0xC00000A2	The target cannot be written to because it is write-protected.
STATUS_INVALID_PARAMETER 0xC000000D	The file name is not a valid parameter.
STATUS_ACCESS_DENIED 0xC0000022	The handle was not opened to write file data or file attributes, or the file has been deleted.
STATUS_PRIVILEGE_NOT_HELD 0xC0000061	The SeRestorePrivilege privilege is not held.
STATUS_SHORT_NAMES_NOT_ENABLED_ON_VOLUME 0xC000019F	Short names are not enabled on this volume.
STATUS_OBJECT_NAME_COLLISION 0xC0000035	The specified name already exists.
STATUS_INFO_LENGTH_MISMATCH 0xC0000004	The specified information record length does not match the length that is required for the specified information class.

2.4.41 FileStandardInformation

Article05/28/2021

This information class is used to query file information.

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

0	1	2	3	4	5	6	7	8	9	10	1	2	3	4	5	6	7	8	9	20	1	2	3	4	5	6	7	8	9	30	1
AllocationSize																															
...																															
EndOfFile																															
...																															
NumberOfLinks																															
DeletePending								Directory								Reserved															

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

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

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

DeletePending (1 byte): A Boolean (section 2.1.8) value. Set to TRUE to indicate that a file deletion has been requested; set to FALSE otherwise.

Directory (1 byte): A Boolean (section 2.1.8) value. Set to TRUE to indicate that the file is a directory; set to FALSE otherwise.

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

This operation returns a status code as specified in section 2.2. Upon success, the status code returned by the function that processes this file information class is STATUS_SUCCESS. The most common error codes are listed in the following table.

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

2.4.48 FileStandardLinkInformation

Article04/07/2025

This information class is used locally to query file link information.<156>

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

Expand table

0	1	2	3	4	5	6	7	8	9	10	1	2	3	4	5	6	7	8	9	20	1	2	3	4	5	6	7	8	9	30	1
NumberOfAccessibleLinks																															
TotalNumberOfLinks																															
DeletePending								Directory								Reserved															

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

TotalNumberOfLinks (4 bytes): A 32-bit unsigned integer that contains the total number of links to this file, including links marked for delete.

DeletePending (1 byte): A Boolean (section 2.1.8) value that MUST be set to TRUE to indicate that a file deletion has been requested; otherwise, FALSE.

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

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

This operation returns a status code as specified in section 2.2. Upon success, the status code returned by the function that processes this file information class is STATUS_SUCCESS. The most common error codes are listed in the following table.

Expand table

Error code	Meaning
STATUS_STATUS_NOT_SUPPORTED 0xC00000BB	The request is not supported.
STATUS_INFO_LENGTH_MISMATCH 0xC0000004	The specified information record length does not match the length that is required for the specified information class.

2.4.49 FileStreamInformation

Article04/07/2025

This information class is used to enumerate the data streams of a file or a directory. A buffer of **FILE_STREAM_INFORMATION** data elements is returned by the server.

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

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

Expand table

0	1	2	3	4	5	6	7	8	9	10	1	2	3	4	5	6	7	8	9	20	1	2	3	4	5	6	7	8	9	30	1
NextEntryOffset																															
StreamNameLength																															
StreamSize																															
...																															
StreamAllocationSize																															
...																															
StreamName (variable)																															
...																															

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

StreamNameLength (4 bytes): A 32-bit unsigned integer that contains the length, in bytes, of the stream name string.

StreamSize (8 bytes): A 64-bit signed integer that contains the size, in bytes, of the stream. The value of this field MUST be greater than or equal to 0x0000000000000000.

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

StreamName (variable): A sequence of Unicode characters containing the name of the stream using the form ":streamname:$DATA", or "::$DATA" for the default data stream, as specified in section 2.1.4. This field is not null-terminated and MUST be handled as a sequence of **StreamNameLength** bytes.

This operation returns a status code as specified in section 2.2. Upon success, the status code returned by the function that processes this file information class is STATUS_SUCCESS. The most common error codes

are listed in the following table.

Expand table

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

2.4.50 FileValidDataLengthInformation

Article04/07/2025

This information class is used to set the valid data length information for a file. A file's valid data length is the length, in bytes, of the data that has been written to the file. This valid data extends from the beginning of the file to the last byte in the file that has not been zeroed or left uninitialized. <157>

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

Expand table

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
ValidDataLength																															
...																															

ValidDataLength (8 bytes): A 64-bit signed integer that contains the new valid data length for the file. This parameter MUST be a positive value that is greater than the current valid data length, but less than or equal to the current file size.

This operation returns a status code as specified in section 2.2. Upon success, the status code returned by the function that processes this file information class is STATUS_SUCCESS. The most common error codes are listed in the following table.

Expand table

Error code	Meaning
STATUS_MEDIA_WRITE_PROTECTED 0xC00000A2	The target cannot be written to because it is write-protected.
STATUS_INVALID_PARAMETER 0xC000000D	The *ValidDataLength* specified is not a valid parameter or the given handle is to a sparse or compressed file.
STATUS_PRIVILEGE_NOT_HELD 0xC0000061	The manage volume privilege is not held.
STATUS_INFO_LENGTH_MISMATCH 0xC0000004	The specified information record length does not match the length that is required for the specified information class.

2.5 File System Information Classes

06/10/2025

File system information classes are numerical values (specified by the Level column in the following table) that specify what information on a particular instance of a file system on a volume is to be queried. File system information classes can retrieve information such as the file system type, volume label, size of the file system, and name of the driver used to access the file system. The table indicates which file system information classes are supported for query and set operations.<158>

Expand table

File system information class	Level	Uses
FileFsVolumeInformation	1	Query
FileFsLabelInformation	2	LOCAL<159>
FileFsSizeInformation	3	Query
FileFsDeviceInformation	4	Query
FileFsAttributeInformation	5	Query
FileFsControlInformation	6	Query, Set
FileFsFullSizeInformation	7	Query
FileFsObjectIdInformation	8	Query, Set
FileFsDriverPathInformation	9	LOCAL<160>
FileFsVolumeFlagsInformation	10	LOCAL<161>
FileFsSectorSizeInformation	11	Query

If an Information Class is specified that does not match the usage in the above table, STATUS_INVALID_INFO_CLASS MUST be returned. If a file system does not implement one of the above defined uses of an Information Class, STATUS_INVALID_PARAMETER MUST be returned.

2.5.1 FileFsAttributeInformation

Article04/07/2025

This information class is used to query attribute information for a file system.

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

Expand table

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 FILE_FILE_COMPRESSION and FILE_VOLUME_IS_COMPRESSED cannot both be set. Any flag values not explicitly mentioned here can be set to any value, and MUST be ignored.<162>

Expand table

Value	Meaning
FILE_SUPPORTS_USN_JOURNAL 0x02000000	The file system implements a USN change journal.
FILE_SUPPORTS_OPEN_BY_FILE_ID 0x01000000	The file system supports opening a file by FileID or ObjectID.
FILE_SUPPORTS_EXTENDED_ATTRIBUTES 0x00800000	The file system persistently stores Extended Attribute information per file.
FILE_SUPPORTS_HARD_LINKS 0x00400000	The file system supports hard linking files.
FILE_SUPPORTS_TRANSACTIONS 0x00200000	The volume supports transactions.<163>
FILE_SEQUENTIAL_WRITE_ONCE 0x00100000	The underlying volume is write once.

Value	Meaning
FILE_READ_ONLY_VOLUME 0x00080000	If set, the volume has been mounted in read-only mode.
FILE_NAMED_STREAMS 0x00040000	The file system supports named streams.
FILE_SUPPORTS_ENCRYPTION 0x00020000	The file system supports the Encrypted File System (EFS). <164>
FILE_SUPPORTS_OBJECT_IDS 0x00010000	The file system supports object identifiers.
FILE_VOLUME_IS_COMPRESSED 0x00008000	The specified volume is a compressed volume. This flag is incompatible with the FILE_FILE_COMPRESSION flag.
FILE_SUPPORTS_POSIX_UNLINK_RENAME 0x00000400	The file system supports POSIX-style delete and rename operations. <165>
FILE_RETURNS_CLEANUP_RESULT_INFO 0x00000200	On a successful cleanup operation, the file system returns information that describes additional actions taken during cleanup, such as deleting the file. File system filters can examine this information in their post-cleanup callback. <166>
FILE_SUPPORTS_REMOTE_STORAGE 0x00000100	The file system supports remote storage. <167>
FILE_SUPPORTS_REPARSE_POINTS 0x00000080	The file system supports reparse points.
FILE_SUPPORTS_SPARSE_FILES 0x00000040	The file system supports sparse files.
FILE_VOLUME_QUOTAS 0x00000020	The file system supports per-user quotas.
FILE_FILE_COMPRESSION 0x00000010	The file volume supports file-based compression. This flag is incompatible with the FILE_VOLUME_IS_COMPRESSED flag.
FILE_PERSISTENT_ACLS 0x00000008	The file system preserves and enforces access control lists (ACLs).
FILE_UNICODE_ON_DISK 0x00000004	The file system supports Unicode in file and directory names. This flag applies only to file and directory names; the file system neither restricts nor interprets the bytes of data within a file.
FILE_CASE_PRESERVED_NAMES	The file system preserves the case of file names when it places a name on disk.

Value	Meaning
0x00000002	
FILE_CASE_SENSITIVE_SEARCH 0x00000001	The file system supports case-sensitive file names when looking up (searching for) file names in a directory.
FILE_SUPPORT_INTEGRITY_STREAMS 0x04000000	The file system supports integrity streams.
FILE_SUPPORTS_BLOCK_REFCOUNTING 0x08000000	The file system supports sharing logical clusters between files on the same volume. The file system reallocates on writes to shared clusters. Indicates that FSCTL_DUPLICATE_EXTENTS_TO_FILE is a supported operation.
FILE_SUPPORTS_SPARSE_VDL 0x10000000	The file system tracks whether each cluster of a file contains valid data (either from explicit file writes or automatic zeros) or invalid data (has not yet been written to or zeroed). File systems that use Sparse VDL do not store a valid data length (section 2.4.50) and do not require that valid data be contiguous within a file.

MaximumComponentNameLength (4 bytes): A 32-bit signed integer that contains the maximum file name component length, in characters, supported by the specified file system. The value of this field MUST be greater than zero and MUST be no more than 255.<168>

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

FileSystemName (variable): A variable-length Unicode field containing the name of the file system. This field is not null-terminated and MUST be handled as a sequence of **FileSystemNameLength** bytes. This field is intended to be informative only. A client SHOULD NOT infer file system type specific behavior from this field.<169>

This operation returns a status code as specified in section 2.2. Upon success, the status code returned by the function that processes this file system information class is STATUS_SUCCESS. The most common error codes are listed in the following table.

Expand table

Error code	Meaning
STATUS_INFO_LENGTH_MISMATCH 0xC0000004	The specified information record length does not match the length that is required for the specified information class.
STATUS_BUFFER_OVERFLOW 0x80000005	The output buffer was filled before all of the file system information could be returned; only a portion of the FileSystemName field is returned.

2.5.2 FileFsControlInformation

Article12/14/2021

This information class is used to query or set quota and content indexing control information for a file system volume.

Setting quota information requires the caller to have permission to open a volume handle or a handle to the quota index file<144> for write access.

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

0	1	2	3	4	5	6	7	8	9	10	1	2	3	4	5	6	7	8	9	20	1	2	3	4	5	6	7	8	9	30	1
FreeSpaceStartFiltering																															
...																															
FreeSpaceThreshold																															
...																															
FreeSpaceStopFiltering																															
...																															
DefaultQuotaThreshold																															
...																															
DefaultQuotaLimit																															
...																															
FileSystemControlFlags																															
Padding																															

FreeSpaceStartFiltering (8 bytes): A 64-bit signed integer that contains the minimum amount of free disk space, in bytes, that is required for the operating system's content indexing service to begin document filtering. This value SHOULD be set to 0, and MUST be ignored.

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.

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.

DefaultQuotaThreshold (8 bytes): A 64-bit unsigned integer that contains the default per-user disk quota warning threshold, in bytes, for the volume. A value of 0xFFFFFFFFFFFFFFFF specifies that no default quota warning threshold per user is set.

DefaultQuotaLimit (8 bytes): A 64-bit unsigned integer that contains the default per-user disk quota limit, in bytes, for the volume. A value of 0xFFFFFFFFFFFFFFFF specifies that no default quota limit per user is set.

FileSystemControlFlags (4 bytes): A 32-bit unsigned integer that contains a bitmask of flags that control quota enforcement and logging of user-related quota events on the volume. The following bit flags are valid in any combination. Bits not defined in the following table SHOULD be set to 0, and MUST be ignored.<145>

Value	Meaning
FILE_VC_CONTENT_INDEX_DISABLED 0x00000008	Content indexing is disabled.
FILE_VC_LOG_QUOTA_LIMIT 0x00000020	An event log entry will be created when the user exceeds the assigned disk quota limit.
FILE_VC_LOG_QUOTA_THRESHOLD 0x00000010	An event log entry will be created when the user exceeds his or her assigned quota warning threshold.
FILE_VC_LOG_VOLUME_LIMIT 0x00000080	An event log entry will be created when the volume's free space limit is exceeded.
FILE_VC_LOG_VOLUME_THRESHOLD 0x00000040	An event log entry will be created when the volume's free space threshold is exceeded.
FILE_VC_QUOTA_ENFORCE 0x00000002	Quotas are tracked and enforced on the volume. Note: FILE_VC_QUOTA_TRACK takes precedence over this flag. In other words, if both FILE_VC_QUOTA_TRACK and FILE_VC_QUOTA_ENFORCE are set, the FILE_VC_QUOTA_ENFORCE flag is ignored. This flag will be ignored if a client attempts to set it.
FILE_VC_QUOTA_TRACK 0x00000001	Quotas are tracked on the volume, but they are not enforced. Tracked quotas enable reporting on the file system space used by system users. If both this flag and FILE_VC_QUOTA_ENFORCE are specified, FILE_VC_QUOTA_ENFORCE is ignored. Note: This flag takes precedence over FILE_VC_QUOTA_ENFORCE. In other words, if both FILE_VC_QUOTA_TRACK and FILE_VC_QUOTA_ENFORCE are set, the FILE_VC_QUOTA_ENFORCE flag is ignored. This flag will be ignored if a client attempts to set it.
FILE_VC_QUOTAS_INCOMPLETE 0x00000100	The quota information for the volume is incomplete because it is corrupt, or the system is in the process of rebuilding the quota information. Note: This does not necessarily imply that FILE_VC_QUOTAS_REBUILDING is set. This flag will be ignored if a client attempts to set it.
FILE_VC_QUOTAS_REBUILDING 0x00000200	The file system is rebuilding the quota information for the volume. Note: This does not necessarily imply that FILE_VC_QUOTAS_INCOMPLETE is set. This flag will be ignored if a client attempts to set it.

Padding (4 bytes): This field SHOULD be set to 0x00000000 and MUST be ignored.

This operation returns a status code as specified in section 2.2. Upon success, the status code returned by the function that processes this file system information class is STATUS_SUCCESS. The most common error codes are listed in the following table.

Error code	Meaning
STATUS_INFO_LENGTH_MISMATCH 0xC0000004	The specified information record length does not match the length that is required for the specified information class.
STATUS_VOLUME_NOT_UPGRADED 0xC000029C	The file system on the volume does not support quotas.

2.5.3 FileFsDriverPathInformation

Article04/07/2025

This information class is used locally to query if a given driver is in the I/O path for a file system volume.

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

Expand table

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
DriverInPath								Reserved																							
DriverNameLength																															
DriverName (variable)																															
...																															

DriverInPath (1 byte): A Boolean (section 2.1.8) value. Set to TRUE if the driver is in the I/O path for the file system volume; set to FALSE otherwise.

Reserved (3 bytes): Reserved for alignment. This field can contain any value and MUST be ignored.

DriverNameLength (4 bytes): A 32-bit unsigned integer that contains the length of the **DriverName** string.

DriverName (variable): A variable-length Unicode field containing the name of the driver for which to query. This sequence of Unicode characters MUST NOT be null-terminated.

This operation returns a status code as specified in section 2.2. Upon success, the status code returned by the function that processes this file system information class is STATUS_SUCCESS. The most common error codes are listed in the following table.

Expand table

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

2.5.4 FileFsFullSizeInformation

Article04/07/2025

This information class is used to query sector size information for a file system volume.

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

Expand table

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.<172>

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.<173>

ActualAvailableAllocationUnits (8 bytes): A 64-bit signed integer that contains the total number of free allocation units on the volume. The value of this field MUST be greater than or equal to 0.

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

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

This operation returns a status code as specified in section 2.2. Upon success, the status code returned by the function that processes this file system information class is STATUS_SUCCESS. The most common error codes are listed in the following table.

Expand table

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

2.5.5 FileFsLabelInformation

Article04/07/2025

This information class is used locally to set the label for a file system volume.

A **FILE_FS_LABEL_INFORMATION** data element, defined as follows, is provided by the caller.

Expand table

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

VolumeLabelLength (4 bytes): A 32-bit unsigned integer that contains the length, in bytes, including the trailing null, if present, of the name for the volume.<174>

VolumeLabel (variable): A variable-length Unicode field containing the name of the volume. The content of this field can be a null-terminated string, or it can be a string padded with the space character to be **VolumeLabelLength** bytes long.

This operation returns a status code as specified in section 2.2. Upon success, the status code returned by the function that processes this file system information class is STATUS_SUCCESS. The most common error codes are listed in the following table.

Expand table

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

2.5.6 FileFsObjectIdInformation

Article04/07/2025

This information class is used to query or set the object ID for a file system data element. The operation MUST fail if the file system does not support object IDs.<175>

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

Expand table

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 (16 bytes)																															
...																															
...																															
ExtendedInfo (48 bytes)																															
...																															
...																															

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.<176>

This operation returns a status code as specified in section 2.2. Upon success, the status code returned by the function that processes this file system information class is STATUS_SUCCESS. The most common error codes are listed in the following table.

Expand table

Error code	Meaning
STATUS_INFO_LENGTH_MISMATCH 0xC0000004	The specified information record length does not match the length that is required for the specified information class.
STATUS_VOLUME_NOT_UPGRADED 0xC000029C	The file system on the volume does not support object IDs.
STATUS_INVALID_PARAMETER 0xC000000D	The file system does not implement object IDs.

2.5.7 FileFsSectorSizeInformation

Article04/07/2025

This information class is used to query for the extended sector size and alignment information for a volume. The message contains a **FILE_FS_SECTOR_SIZE_INFORMATION** data element.<177>

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

Expand table

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
LogicalBytesPerSector																															
PhysicalBytesPerSectorForAtomicity																															
PhysicalBytesPerSectorForPerformance																															
FileSystemEffectivePhysicalBytesPerSectorForAtomicity																															
Flags																															
ByteOffsetForSectorAlignment																															
ByteOffsetForPartitionAlignment																															

LogicalBytesPerSector (4 bytes): A 32-bit unsigned integer that contains the number of bytes in a logical sector for the device backing the volume. This field is the unit of logical addressing for the device and is not the unit of atomic write. Applications SHOULD NOT utilize this value for operations requiring physical sector alignment.

PhysicalBytesPerSectorForAtomicity (4 bytes): A 32-bit unsigned integer that contains the number of bytes in a physical sector for the device backing the volume. Note that this is the reported physical sector size of the device and is the unit of atomic write. Applications SHOULD<178> utilize this value for operations requiring sector alignment.

PhysicalBytesPerSectorForPerformance (4 bytes): A 32-bit unsigned integer that contains the number of bytes in a physical sector for the device backing the volume. This is the reported physical sector size of the device and is the unit of performance. Applications SHOULD<179> utilize this value for operations requiring sector alignment.

FileSystemEffectivePhysicalBytesPerSectorForAtomicity (4 bytes): A 32-bit unsigned integer containing the unit, in bytes, that the file system on the volume will use for internal operations that require alignment and atomicity.<180>

Flags (4 bytes): A 32-bit unsigned integer that indicates the flags for this operation. Currently defined flags are:

Expand table

Value	Meaning
SSINFO_FLAGS_ALIGNED_DEVICE 0x00000001	When set, this flag indicates that the first physical sector of the device is aligned with the first logical sector. When not set, the first physical sector of the device is misaligned with the first logical sector.
SSINFO_FLAGS_PARTITION_ALIGNED_ON_DEVICE 0x00000002	When set, this flag indicates that the partition is aligned to physical sector boundaries on the storage device.
SSINFO_FLAGS_NO_SEEK_PENALTY 0x00000004	When set, the device reports that it does not incur a seek penalty (this typically indicates that the device does not have rotating media, such as flash-based disks).
SSINFO_FLAGS_TRIM_ENABLED 0x00000008	When set, the device supports TRIM operations, either T13 (ATA) TRIM or T10 (SCSI/SAS) UNMAP.

ByteOffsetForSectorAlignment (4 bytes): A 32-bit unsigned integer that contains the logical sector offset within the first physical sector where the first logical sector is placed, in bytes. If this value is set to SSINFO_OFFSET_UNKNOWN (0XFFFFFFFF), there was insufficient information to compute this field.<181>

ByteOffsetForPartitionAlignment (4 bytes): A 32-bit unsigned integer that contains the byte offset from the first physical sector where the first partition is placed. If this value is set to SSINFO_OFFSET_UNKNOWN (0XFFFFFFFF), there was either insufficient information or an error was encountered in computing this field.

This operation returns a status code as specified in section 2.2. Upon success, the status code returned by the function that processes this file system information class is STATUS_SUCCESS. The most common error codes are listed in the following table.

Expand table

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

2.5.8 FileFsSizeInformation

Article04/07/2025

This information class is used to query sector size information for a file system volume.

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

Expand table

0	1	2	3	4	5	6	7	8	9	10	1	2	3	4	5	6	7	8	9	20	1	2	3	4	5	6	7	8	9	30	1
TotalAllocationUnits																															
...																															
AvailableAllocationUnits																															
...																															
SectorsPerAllocationUnit																															
BytesPerSector																															

TotalAllocationUnits (8 bytes): A 64-bit signed integer that contains the total number of allocation units on the volume that are available to the user associated with the calling thread. This value MUST be greater than or equal to 0.<182>

AvailableAllocationUnits (8 bytes): A 64-bit signed integer that contains the total number of free allocation units on the volume that are available to the user associated with the calling thread. This value MUST be greater than or equal to 0.<183>

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

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

This operation returns a status code as specified in section 2.2. Upon success, the status code returned by the function that processes this file system information class is STATUS_SUCCESS. The most common error codes are listed in the following table.

Expand table

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

2.5.9 FileFsVolumeInformation

Article04/07/2025

This information class is used to query information on a volume on which a file system is mounted.

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

Expand table

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
VolumeCreationTime																															
...																															
VolumeSerialNumber																															
VolumeLabelLength																															
SupportsObjects									Reserved																						
VolumeLabel (variable)																															
...																															

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

VolumeSerialNumber (4 bytes): A 32-bit unsigned integer that contains the serial number of the volume. The serial number is an opaque value generated by the file system at format time, and is not necessarily related to any hardware serial number for the device on which the file system is located. No specific format or content of this field is required for protocol interoperation. This value is not required to be unique.

VolumeLabelLength (4 bytes): A 32-bit unsigned integer that contains the length, in bytes, including the trailing null, if present, of the name of the volume.<184>

SupportsObjects (1 byte): A Boolean (section 2.1.8) value. Set to TRUE if the file system supports object-oriented file system objects; set to FALSE otherwise.<185>

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

VolumeLabel (variable): A variable-length Unicode field containing the name of the volume. The content of this field can be a null-terminated string or can be a string padded with the space character to be **VolumeLabelLength** bytes long.

This operation returns a status code as specified in section 2.2. Upon success, the status code returned by the function that processes this file system information class is STATUS_SUCCESS. The most common error codes are listed in the following table.

If the volume label is greater than 32 characters, return the first 32 characters of the label and STATUS_SUCCESS.

Expand table

Error code	Meaning
STATUS_INFO_LENGTH_MISMATCH 0xC0000004	The specified information record length does not match the length that is required for the specified information class.
STATUS_BUFFER_OVERFLOW 0x80000005	The output buffer was filled before all of the volume information could be returned; only a portion of the **VolumeLabel** field is returned.

2.5.10 FileFsDeviceInformation

Article04/07/2025

This information class is used to query device information associated with a file system volume.

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

Expand table

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
DeviceType																															
Characteristics																															

DeviceType (4 bytes): This identifies the type of given volume. It MUST be one of the following.

Expand table

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.

Expand table

Value	Meaning
FILE_REMOVABLE_MEDIA 0x00000001	Indicates that the storage device supports removable media. Notice that this characteristic indicates removable media, not a removable device. For example, drivers for JAZ drive devices specify this characteristic, but drivers for PCMCIA flash disks do not.
FILE_READ_ONLY_DEVICE 0x00000002	Indicates that the device cannot be written to.
FILE_FLOPPY_DISKETTE 0x00000004	Indicates that the device is a floppy disk device.
FILE_WRITE_ONCE_MEDIA 0x00000008	Indicates that the device supports write-once media.

Value	Meaning
FILE_REMOTE_DEVICE 0x00000010	Indicates that the volume is for a remote file system like SMB or CIFS.
FILE_DEVICE_IS_MOUNTED 0x00000020	Indicates that a file system is mounted on the device.
FILE_VIRTUAL_VOLUME 0x00000040	Indicates that the volume does not directly reside on storage media but resides on some other type of media (memory for example).
FILE_DEVICE_SECURE_OPEN 0x00000100	By default, volumes do not check the ACL associated with the volume, but instead use the ACLs associated with individual files on the volume. When this flag is set the volume ACL is also checked.
FILE_CHARACTERISTIC_TS_DEVICE 0x00001000	Indicates that the device object is part of a Terminal Services device stack. See [MS-RDPBCGR] for more information.
FILE_CHARACTERISTIC_WEBDAV_DEVICE 0x00002000	Indicates that a web-based Distributed Authoring and Versioning (WebDAV) file system is mounted on the device. See [MS-WDVME] for more information.
FILE_DEVICE_ALLOW_APPCONTAINER_TRAVERSAL 0x00020000	The IO Manager normally performs a full security check for traverse access on every file open when the client is an appcontainer. Setting of this flag bypasses this enforced traverse access check if the client token already has traverse privileges. <186>
FILE_PORTABLE_DEVICE 0x0004000	Indicates that the given device resides on a portable bus like USB or Firewire and that the entire device (not just the media) can be removed from the system.

This operation returns a status code as specified in section 2.2. Upon success, the status code returned by the function that processes this file system information class is STATUS_SUCCESS. The most common error codes are listed in the following table.

Expand table

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

2.6 File Attributes

Article04/07/2025

The following attributes are defined for files and directories. They can be used in any combination unless noted in the description of the attribute's meaning. There is no file attribute with the value 0x00000000 because a value of 0x00000000 in the **FileAttributes** field means that the file attributes for this file MUST NOT be changed when setting basic information for the file.

Note: File systems silently ignore any attribute that is not supported by that file system. Unsupported attributes MUST NOT be persisted on the media. It is recommended that unsupported attributes be masked off when encountered.

Expand table

Value	Meaning
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_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_SYSTEM 0x00000004	A file or directory that the operating system uses a part of or uses exclusively.
FILE_ATTRIBUTE_DIRECTORY 0x00000010	This item is a directory.
FILE_ATTRIBUTE_ARCHIVE 0x00000020	A file or directory that requires to be archived. Applications use this attribute to mark files for backup or removal.
FILE_ATTRIBUTE_NORMAL 0x00000080	A file that does not have other attributes set. This flag is used to clear all other flags by specifying it with no other flags set. This flag MUST be ignored if other flags are set.<187>
FILE_ATTRIBUTE_TEMPORARY 0x00000100	A file that is being used for temporary storage. The operating system can 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.

Value	Meaning
FILE_ATTRIBUTE_SPARSE_FILE 0x00000200	A file that is a sparse file.
FILE_ATTRIBUTE_REPARSE_POINT 0x00000400	A file or directory that has an associated reparse point.
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_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_NOT_CONTENT_INDEXED 0x00002000	A file or directory that is not indexed by the content indexing service.
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_INTEGRITY_STREAM 0x00008000	A file or directory that is configured with integrity support. For a file, all data streams in the file have integrity support. For a directory, integrity support is the default for newly created files and subdirectories, unless the caller specifies otherwise.<188>
FILE_ATTRIBUTE_NO_SCRUB_DATA 0x00020000	A file or directory that is configured to be excluded from the data integrity scan. For a directory configured with FILE_ATTRIBUTE_NO_SCRUB_DATA, the default for newly created files and subdirectories is to inherit the **FILE_ATTRIBUTE_NO_SCRUB_DATA** attribute.<189>
FILE_ATTRIBUTE_RECALL_ON_OPEN 0x00040000	This attribute appears only in directory enumeration classes (FILE_DIRECTORY_INFORMATION, FILE_BOTH_DIR_INFORMATION, etc.). When this attribute is set, it means that the file or directory has no physical representation on the local system; the item is virtual. Opening the item will be more expensive than usual because it will cause at least some of the file or directory content to be fetched from a remote store. This attribute can only be set by kernel-mode components. This attribute is for use with hierarchical storage management software. <190>

Value	Meaning
FILE_ATTRIBUTE_PINNED 0x00080000	This attribute indicates user intent that the file or directory should be kept fully present locally even when not being actively accessed. This attribute is for use with hierarchical storage management software.<191>
FILE_ATTRIBUTE_UNPINNED 0x00100000	This attribute indicates that the file or directory should not be kept fully present locally except when being actively accessed. This attribute is for use with hierarchical storage management software.<192>
FILE_ATTRIBUTE_RECALL_ON_DATA_ACCESS 0x00400000	When this attribute is set, it means that the file or directory is not fully present locally. For a file this means that not all of its data is on local storage (for example, it may be sparse with some data still in remote storage). For a directory it means that some of the directory contents are being virtualized from another location. Reading the file or enumerating the directory will be more expensive than usual because it will cause at least some of the file or directory content to be fetched from a remote store. Only kernel-mode callers can set this attribute. This attribute is for use with hierarchical storage management software. <193>

2.7 Directory Change Notifications

Article02/14/2019

The following definitions are part of the Directory Change Notification algorithm defined in [MS-FSA] section 2.1.5.10.

2.7.1 FILE_NOTIFY_INFORMATION

Article09/20/2023

The **FILE_NOTIFY_INFORMATION** structure contains the changes for which the client is being notified. The structure consists of the following.

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																															
Action																															
FileNameLength																															
FileName (variable)																															

NextEntryOffset (4 bytes): The offset, in bytes, from the beginning of this structure to the subsequent **FILE_NOTIFY_INFORMATION** structure. If there are no subsequent structures, the **NextEntryOffset** field MUST be 0. **NextEntryOffset** MUST always be an integral multiple of 4. The **FileName** array MUST be padded to the next 4-byte boundary counted from the beginning of the structure.

Action (4 bytes): The changes that occurred on the file. This field MUST contain one of the following values.<169>

Value	Meaning
FILE_ACTION_ADDED 0x00000001	The file was renamed, and **FileName** contains the new name. This notification is only sent when the rename operation changes the directory the file resides in. The client will also receive a FILE_ACTION_REMOVED notification. This notification will not be received if the file is renamed within a directory.
FILE_ACTION_REMOVED 0x00000002	The file was renamed, and **FileName** contains the old name. This notification is only sent when the rename operation changes the directory the file resides in. The client will also receive a FILE_ACTION_ADDED notification. This notification will not be received if the file is renamed within a directory.
FILE_ACTION_MODIFIED 0x00000003	The file was modified. This can be a change to the data or attributes of the file.
FILE_ACTION_RENAMED_OLD_NAME 0x00000004	The file was renamed, and **FileName** contains the old name. This notification is only sent when the rename operation does not change the directory the file resides in. The client will also receive a FILE_ACTION_RENAMED_NEW_NAME notification. This notification will not be received if the file is renamed to a different directory.
FILE_ACTION_RENAMED_NEW_NAME 0x00000005	The file was renamed, and **FileName** contains the new name. This notification is only sent when the rename operation does not change the directory the file resides in. The client will also receive a FILE_ACTION_RENAMED_OLD_NAME notification. This notification will not be received if the file is renamed to a different directory.
FILE_ACTION_ADDED_STREAM	The file was added to a named stream.

Value	Meaning
0x00000006	
FILE_ACTION_REMOVED_STREAM 0x00000007	The file was removed from the named stream.
FILE_ACTION_MODIFIED_STREAM 0x00000008	The file was modified. This can be a change to the data or attributes of the file.
FILE_ACTION_REMOVED_BY_DELETE 0x00000009	An object ID was removed because the file the object ID referred to was deleted. This notification is only sent when the directory being monitored is the special directory "\$Extend\$ObjId:$O:$INDEX_ALLOCATION".
FILE_ACTION_ID_NOT_TUNNELLED 0x0000000A	An attempt to tunnel object ID information to a file being created or renamed failed because the object ID is in use by another file on the same volume. This notification is only sent when the directory being monitored is the special directory "\$Extend\$ObjId:$O:$INDEX_ALLOCATION".
FILE_ACTION_TUNNELLED_ID_COLLISION 0x0000000B	An attempt to tunnel object ID information to a file being renamed failed because the file already has an object ID. This notification is only sent when the directory being monitored is the special directory "\$Extend\$ObjId:$O:$INDEX_ALLOCATION".

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

FileNameLength (4 bytes): The length, in bytes, of the file name in the **FileName** field.

FileName (variable): A Unicode string with the name of the file that changed.

2.8 Cluster Shared Volume File System IOCTLs

Article04/07/2025

SQL Server Remote Storage Profile [MS-SQLRS] relies on the I/O control (IOCTL) code structures, and definitions in this section, to interpret certain fields that can be sent or received as part of its processing. See section 2.3 for more information about processing.

2.8.1 IOCTL_STORAGE_QUERY_PROPERTY Request

Article10/30/2020

The IOCTL_STORAGE_QUERY_PROPERTY Request message requests that the server return the properties of a storage device or verify that the request is supported.

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
PropertyId																															
QueryType																															

PropertyId (4 bytes): This field MUST be set to 0x00000006.

QueryType (4 bytes): Contains flags indicating the type of query to be performed.

Value	Meaning
0x00000000 PropertyStandardQuery	Query to return the IOCTL_STORAGE_QUERY_PROPERTY Reply message.
0x00000001 PropertyExistsQuery	Query to see whether **PropertyId** is supported.

2.8.2 IOCTL_STORAGE_QUERY_PROPERTY Reply

Article02/14/2019

The IOCTL_STORAGE_QUERY_PROPERTY Reply message contains the storage alignment information.

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
Version																															
Size																															
BytesPerCacheLine																															
BytesOffsetForCacheAlignment																															
BytesPerLogicalSector																															
BytesPerPhysicalSector																															
BytesOffsetForSectorAlignment																															

Version (4 bytes): Contains the size of this structure, in bytes.

Size (4 bytes): Specifies the total size of the data returned, in bytes.

BytesPerCacheLine (4 bytes): The number of bytes in a cache line of the device.

BytesOffsetForCacheAlignment (4 bytes): The address offset necessary for proper cache access alignment, in bytes.

BytesPerLogicalSector (4 bytes): The number of bytes in a logical sector of the device.

BytesPerPhysicalSector (4 bytes): The number of bytes in a physical sector of the device.

BytesOffsetForSectorAlignment (4 bytes): The logical sector offset within the first physical sector where the first logical sector is placed, in bytes.

2.8.3 IOCTL_VOLUME_GET_GPT_ATTRIBUTES Request

Article02/14/2019

The IOCTL_VOLUME_GET_GPT_ATTRIBUTES Request message retrieves the attributes for a volume.

This message does not contain any additional data elements.

2.8.4 IOCTL_VOLUME_GET_GPT_ATTRIBUTES Reply

Article10/30/2020

The IOCTL_VOLUME_GET_GPT_ATTRIBUTES Reply message returns the attributes of the volume.

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
GptAttributes																															
...																															

GptAttributes (4 bytes): Specifies all of the attributes associated with a volume.

Value	Meaning
GPT_BASIC_DATA_ATTRIBUTE_READ_ONLY 0x1000000000000000	The volume is read-only.
GPT_BASIC_DATA_ATTRIBUTE_SHADOW_COPY 0x2000000000000000	The volume is a shadow copy of another volume.
GPT_BASIC_DATA_ATTRIBUTE_HIDDEN 0x4000000000000000	The volume is hidden.
GPT_BASIC_DATA_ATTRIBUTE_NO_DRIVE_LETTER 0x8000000000000000	The volume is not assigned a default drive letter.

3 Structure Examples

Article02/14/2019

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

Article04/23/2024

4.1 Security Considerations for Implementers

Article02/14/2019

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 can 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

Article04/23/2024

None.

5 Appendix A: NTFS Alternate Streams

Article04/23/2024

5.1 NTFS Streams

All files on an NTFS volume consist of at least one stream - the main stream – this is the normal, viewable file in which data is stored. The full name of a stream is of the form below.

<filename>:<stream name>:<stream type>

The default data stream has no name. That is, the fully qualified name for the default stream for a file called "sample.txt" is "sample.txt::$DATA" since "sample.txt" is the name of the file and "$DATA" is the stream type.

A user can create a named stream in a file and "$DATA" as a legal name. That means that for this stream, the full name is sample.txt:$DATA:$DATA. If the user had created a named stream of name "bar", its full name would be sample.txt:bar:$DATA. Any legal characters for a file name are legal for the stream name (including spaces). For more information about the naming format for streams, see [MS-FSCC]. For more information about the attributes of a stream, see [MS-FSA].

In the case of directories, there is no default data stream, but there is a default directory stream. Directories are the stream type $INDEX_ALLOCATION. The default stream name for the type $INDEX_ALLOCATION (a directory stream) is $I30. (This contrasts with the default stream name for a $DATA stream, which has an empty stream name.) The following are equivalent:

Dir C:\Users

Dir C:\Users:$I30:$INDEX_ALLOCATION

Dir C:\Users::$INDEX_ALLOCATION

Although directories do not have a default data stream, they can have named data streams. These alternate data streams are not normally visible, but can be observed from a command line using the /R option of the DIR command.

Last updated on 11/21/2025

5.2 NTFS Attribute Types

Article09/20/2023

On a NTFS volume, each unit of information associated with a file including its name, its owner, its timestamp, its contents, and so on, is implemented as a file attribute. A file's data is an attribute; the "Data Attribute" known as $DATA. A number of attributes exist on a NTFS volume. The attribute names used by NTFS are listed in the table below.

Attribute Name	Description
$ATTRIBUTE_LIST	Lists the location of all attribute records that do not fit in the MFT record
$BITMAP	Attribute for Bitmaps
$DATA	Contains the default file data
$EA	Extended the attribute index
$EA_INFORMATION	Extended attribute information
$FILE_NAME	File name
$INDEX_ALLOCATION	The type name for a Directory Stream. A string for the attribute code for index allocation
$INDEX_ROOT	Used to support folders and other indexes
$LOGGED_UTILITY_STREAM	Use by the encrypting file system
$OBJECT_ID	Unique GUID for every MFT record
$PROPERTY_SET	Obsolete
$REPARSE_POINT	Used for volume mount points
$SECURITY_DESCRIPTOR	Security descriptor stores ACL and SIDs
$STANDARD_INFORMATION	Standard information, such as file times and quota data
$SYMBOLIC_LINK	Obsolete
$TXF_DATA	Transactional NTFS data
$VOLUME_INFORMATION	Version and state of the volume
$VOLUME_NAME	Name of the volume
$VOLUME_VERSION	Obsolete. Volume version

A comprehensive discussion and explanation about attributes is available in [WININTERNALS] Chapter 12 and [MSFT-NTFSWorks].

5.3 NTFS Reserved File Names

Article09/20/2023

NTFS uses a number of names as part of the file system internals. The names used by NTFS within the root directory are listed in the following table:

Filename	Description
\$Mft	Master File Table (MFT) - an index of every file
\$MftMirr	A backup copy of the first 4 records of the MFT
\$LogFile	Transactional logging file
\$Volume	Serial number, creation time, dirty flag
\$AttrDef	Attribute definitions
\$Bitmap	Contains the volume's cluster map (in-use vs. free)
\$Boot	Boot record of the volume
\$BadClus	Lists bad clusters on the volume
\$Secure	Security descriptors used by the volume
\$UpCase	Table of uppercase characters used for collating
\$Extend	A directory

An additional set of names are found in the system directory as follows:

Filename	Description
\$Extend\$Config	Use for NTFS repair activity
\$Extend\$Delete	Delete file name
\$Extend\$ObjId	Unique Ids given to every file
\$Extend\$Quota	Quota information
\$Extend\$Repair	Repair name
\$Extend\$Repair.log	Repair log name
\$Extend\$Reparse	Reparse point information
\$Extend\$RmMetadata	Transactional NTFS resource manager metadata name

Filename	Description
\$Extend\$Tops	Transactional NTFS Old Page Stream, used to store data that has been overwritten inside a currently active transaction
\$Extend\$Txf	Transactional NTFS
\$Extend\$TxfLog	Transactional NTFS log

5.4 NTFS Stream Names

Article09/20/2023

NTFS by convention uses names starting with '$' for internal metadata files and streams on those internal metadata files. There is no mechanism to stop applications from using names of this form; therefore, it is recommended that names of this form not be used internally by an object store for a server environment except when emulating NTFS metadata streams such as "\$Extend\$Quota:$Q:$INDEX_ALLOCATION" or "\$Extend\$Reparse:$R:$INDEX_ALLOCATION".

Stream Names currently used by NTFS are as follows:

NTFS Internal Stream Names	Example
$I30	Default name for directory streams C:\Users:$I30:$INDEX_ALLOCATION
$O	\$Extend\$ObjId:$O:$INDEX_ALLOCATION
$Q	\$Extend\$Quota:$Q:$INDEX_ALLOCATION
$R	\$Extend\$Reparse:$R:$INDEX_ALLOCATION
$J	\$Extend\$UsnJrnl:$J:$DATA
$MAX	\$Extend\$UsnJrnl:$MAX:$DATA
$SDH	\$Secure:$SDH:$INDEX_ALLOCATION
$SII	\$Secure:$SII:$INDEX_ALLOCATION

5.5 NTFS Stream Types

Article10/30/2020

Names currently used are as follows:

NTFS Stream Types
$DATA
$INDEX_ALLOCATION
$BITMAP

5.6 Known Alternate Stream Names

Article04/07/2025

Selection of an alternate stream name, is in principle, identical to selection of a filename. An application might need to check whether a name is in use prior to attempting to use a name. When an application has successfully avoided a file name conflict, it has complete control over any stream names that it might wish to use. It is advisable to use textual GUID (GUIDString) as stream names in order to avoid conflicts. Injection of streams into files that an application does not completely own has the potential to cause unpredictable behavior and can be flagged by virus scanning software.

5.6.1 Zone.Identifier Stream Name

Article02/10/2025

Windows Internet Explorer uses the stream name Zone.Identifier for storage of URL security zones.

The fully qualified form is sample.txt: Zone.Identifier:$DATA

The stream is a simple text stream of the form:

[ZoneTransfer]

ZoneId=3

[MSDN-SECZONES] gives an explanation of security zones.

5.6.2 Outlook Express Properties Stream Name

Article02/14/2019

Outlook Express uses the stream name OECustomProperty for storage of custom properties related to email files.

The fully qualified form is sample.eml:OECustomProperty:$DATA

5.6.3 Document Properties Stream Name

Property sets, when applied to files, use a number of different stream names. The initial character is Unicode U+2663, known as (BLACK CLUB).

The names "♣BnhqlkugBim0elg1M1pt2tjdZe", "♣SummaryInformation" and the GUID {4c8cc155-6c1e-11d1-8e41-00c04fb9386d} are used.

The fully qualified names would be as follows:

sample.doc:♣BnhqlkugBim0elg1M1pt2tjdZe:$DATA

sample.doc:♣SummaryInformation:$DATA

sample.gif:{4c8cc155-6c1e-11d1-8e41-00c04fb9386d}:$DATA

Last updated on 11/21/2025

5.6.4 Encryptable Thumbnails Stream Name

Article02/14/2019

Windows Shell uses the stream name "encryptable" to store attributes relating to thumbnails in the thumbnails database.

The fully qualified name would be as follows:

Thumbs.db:encryptable:$DATA

5.6.5 Internet Explorer Favicon Stream Name

Article02/14/2019

Windows Internet Explorer uses the stream name "favicon" for storing favorite ICONs for web pages.

The fully qualified name would be as follows:

Pages.url:favicon:$DATA

5.6.6 Macintosh Supported Stream Names

Article02/14/2019

Two stream names exist for compatibility with Macintosh operating system property lists. These names are "AFP_AfpInfo" and "AFP_Resource".

The fully qualified name would be as follows:

Sample.txt:AFP_AfpInfo:$DATA

Sample.txt:AFP_Resource:$DATA

5.6.7 XPRESS Stream Name

Article02/14/2019

The stream name "{59828bbb-3f72-4c1b-a420-b51ad66eb5d3}.XPRESS" is used during remote differential compression.

The fully qualified name would be as follows:

Sample.bin: {59828bbb-3f72-4c1b-a420-b51ad66eb5d3}.XPRESS:$DATA

6 Appendix B: Product Behavior

Article04/07/2025

The information in this specification is applicable to the following Microsoft products or supplemental software. References to product versions include updates to those products.

The terms "earlier" and "later", when used with a product version, refer to either all preceding versions or all subsequent versions, respectively. The term "through" refers to the inclusive range of versions. Applicable Microsoft products are listed chronologically in this section.

- Windows NT 4.0 operating system
- Windows 98 operating system
- Windows 98 operating system Second Edition
- Windows 2000 operating system
- Windows XP operating system
- Windows Server 2003 operating system
- Windows Vista operating system
- Windows Server 2008 operating system
- Windows 7 operating system
- Windows Server 2008 R2 operating system
- Windows 8 operating system
- Windows Server 2012 operating system
- Windows 8.1 operating system
- Windows Server 2012 R2 operating system
- Windows 10 operating system
- Windows Server 2016 operating system
- Windows Server 2019 operating system
- Windows Server 2022 operating system
- Windows 11 operating system

- Windows Server 2025 operating system

Exceptions, if any, are noted in this section. If an update version, service pack or Knowledge Base (KB) number appears with a product name, the behavior changed in that update. The new behavior also applies to subsequent updates unless otherwise specified. If a product edition appears with the product version, behavior is different in that product edition.

Unless otherwise specified, any statement of optional behavior in this specification that is prescribed using the terms "SHOULD" or "SHOULD NOT" implies product behavior in accordance with the SHOULD or SHOULD NOT prescription. Unless otherwise specified, the term "MAY" implies that the product does not follow the prescription.

<1> Section 2.1.2.1: All reparse tags defined by Microsoft components MUST have the high bit set to 1. Non-Microsoft reparse tags MUST have the high bit set to 0.

<2> Section 2.1.2.1: These are Microsoft reparse tags. Except where explicitly allowed, clients MUST NOT process the Microsoft reparse tag data buffers.

<3> Section 2.1.2.1: The Windows Home Server Drive Extender is part of the Windows Home Server product.

<4> Section 2.1.2.1: The filter manager test harness is not shipped with Windows.

<5> Section 2.1.3.1: When a file is moved or copied from one volume to another, the **ObjectId** member value changes to a random unique value to avoid the potential for **ObjectId** collisions because the object ID is not guaranteed to be unique across volumes.

<6> Section 2.1.3.1: The NTFS file system places no constraints on the format of the 48 bytes of information following the ObjectId in this structure. This format of the FILE_OBJECTID_BUFFER is used on Windows by the Microsoft Distributed Link Tracking Service (see [MS-DLTW] section 3.1.6).

<7> Section 2.1.3.2: Windows places Distributed Link Tracking (DLT) information into the ExtendedInfo field for use by the Distributed Link Tracking (DLT) protocols (see [MS-DLTW] section 3.1.6).

<8> Section 2.1.4: The following Windows file systems provide alternate data stream functionality: NTFS, ReFS and Universal Disk Format (UDF). ReFS supports alternate data streams of up to 128 KB in length in Windows 8.1 and later. ReFS does not support renaming of alternate data streams.

<9> Section 2.1.8: Windows defines a TRUE as "1"; however, it will interpret any nonzero value as TRUE.

<10> Section 2.1.9: The following table lists the file systems that support the **64-bit file ID**:

Expand table

64 bit file ID	Generate	Stable	Unique
FAT	Yes	No	No
EXFAT	Yes	No	No
FAT32	Yes	No	No
Cdfs	No	n/a	n/a
UDFS	Yes	Yes	Yes
NTFS	Yes	Yes	Yes
ReFS	Yes	Yes	Yes

NTFS computes the **64-bit file ID** as follows: the low 48 bits are the index of the file's primary record in the master file table (MFT); the remaining 16 bits are a sequence number. Therefore, it is possible, though rare, that a different file can have the same 64-bit file ID as a file on that volume had in the past.

ReFS maps a subset of the possible **128-bit file ID** values to a 64-bit value using a reversible algorithm; for values outside of this subset, ReFS sets the **64-bit file ID** to -1.

<11> Section 2.1.10: The following table lists the file systems that support the 128-bit file ID:

Expand table

128 bit file ID	Generate	Stable	Unique
FAT	No	n/a	n/a
EXFAT	No	n/a	n/a
FAT32	No	n/a	n/a
Cdfs	No	n/a	n/a
UDFS	No	n/a	n/a
NTFS	Yes	Yes	Yes
ReFS	Yes	Yes	Yes

NTFS computes the **128-bit file ID** as follows: the low 48 bits are the index of the file's primary record in the master file table (MFT), the next 16 bits are a sequence number, and the high 64

bits MUST be zero. Therefore, it is possible, though rare, that a different file can have the same **128-bit file ID** as a file on that volume had in the past.

ReFS computes the **128-bit file ID** as follows: the low 64 bits consists of an index uniquely identifying the file's parent directory on the volume. The high 64-bits consists of an index uniquely identifying the file within that directory.

<12> Section 2.1.11: The Token is defined in [INCITS-T10/11-059].

<13> Section 2.1.11: When provided by a client to a server for an FSCTL_OFFLOAD_WRITE operation, this Token value requests that the server logically write zeros.

<14> Section 2.2: NTFS supports reparse points, object IDs, and the update sequence number (USN) change journal; ReFS supports reparse points and the USN change journal. The Microsoft FAT, EXFAT, CDFS, and UDFS file systems do not support these attributes. Therefore, FSCTLs involving these technologies will return STATUS_INVALID_DEVICE_REQUEST when the specified file or directory is located on a volume formatted with the FAT file system. Windows also returns STATUS_INVALID_DEVICE_REQUEST when a required file system filter is supported by the file system but is not installed (see section 2.3.90).

<15> Section 2.2: The following table lists FSCTLs that are not supported remotely and that, if received by the object store, will respond with a status code other than STATUS_INVALID_DEVICE_REQUEST, as specified in section 2.2.

Expand table

FSCTL name	FSCTL function number	Status Code
FSCTL_GET_BOOT_AREA_INFO	0x90230	STATUS_INVALID_PARAMETER
FSCTL_GET_RETRIEVAL_POINTER_BASE	0x90234	STATUS_INVALID_PARAMETER
FSCTL_IS_VOLUME_DIRTY	0x90078	STATUS_INVALID_PARAMETER
FSCTL_ALLOW_EXTENDED_DASD_IO	0x90083	STATUS_ACCESS_DENIED
FSCTL_LOOKUP_STREAM_FROM_CLUSTER	0x901FC	STATUS_INVALID_PARAMETER
FSCTL_EXTEND_VOLUME	0x900F0	STATUS_INVALID_PARAMETER
FSCTL_SHRINK_VOLUME	0x901B0	STATUS_INVALID_PARAMETER
FSCTL_FILE_PREFETCH	0x90120	STATUS_INVALID_PARAMETER
FSCTL_SET_PERSISTENT_VOLUME_STATE	0x90238	STATUS_INVALID_PARAMETER
FSCTL_QUERY_PERSISTENT_VOLUME_STATE	0x9023C	STATUS_INVALID_PARAMETER

FSCTL name	FSCTL function number	Status Code
FSCTL_SD_GLOBAL_CHANGE	0x901F4	STATUS_INVALID_PARAMETER

<16> 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.

<17> Section 2.3.2: Windows will try 16 times to generate a unique ID, and will fail with this status if 16 attempts have been unsuccessful.

<18> Section 2.3.7: FSCTL_DUPLICATE_EXTENTS_TO_FILE is only supported by the ReFS file system in Windows Server 2016 and later.

<19> Section 2.3.8: FSCTL_DUPLICATE_EXTENTS_TO_FILE is only supported by the ReFS file system in Windows Server 2016 and later.

<20> Section 2.3.8: Applicable Windows Server releases return STATUS_INVALID_HANDLE if the source file handle is closed, and STATUS_FILE_CLOSED if the target file handle is closed.

<21> Section 2.3.9: FSCTL_DUPLICATE_EXTENTS_TO_FILE_EX request is only supported by the ReFS file system in Windows 10 v1803 operating system and Windows Server v1803 operating system.

<22> Section 2.3.10: FSCTL_DUPLICATE_EXTENTS_TO_FILE_EX reply is only supported by the ReFS file system in Windows 10 v1803 and Windows Server v1803.

<23> Section 2.3.11: This FSCTL is implemented on ReFS, NTFS, FAT, and exFAT file systems. Other file systems return STATUS_INVALID_DEVICE_REQUEST.

<24> Section 2.3.12: This FSCTL is implemented on ReFS, NTFS, FAT, and exFAT file systems. Other file systems return STATUS_INVALID_DEVICE_REQUEST.

<25> Section 2.3.16: NTFS always returns at least 2 bytes and up to 8 bytes of trailing padding after each entry in the reply, including the last entry.

<26> Section 2.3.18: The LZNT1 is the only compression algorithm implemented on Windows 2000, Windows XP, Windows Server 2003, Windows Vista, Windows Server 2008, Windows 7, and Windows Server 2008 R2.

<27> Section 2.3.18: Windows 2000, Windows XP, Windows Server 2003, Windows Vista, Windows Server 2008, Windows 7, and Windows Server 2008 R2 support file compression on volumes that are formatted with the NTFS file system and have a cluster size less than or equal to 4 kilobytes.

<28> Section 2.3.19: The FSCTL_GET_INTEGRITY_INFORMATION_Request (section 2.3.19) message is supported only by the ReFS file system.

<29> Section 2.3.28:

- Windows NT 4.0 returns STATUS_INVALID_DEVICE_REQUEST for a file on an NTFS, FAT, or CDFS file system.
- Windows 2000 returns STATUS_INVALID_DEVICE_REQUEST for a file on a FAT or CDFS file system.
- Windows XP returns STATUS_INVALID_DEVICE_REQUEST for a file on a FAT or CDFS file system.
- Windows Server 2003 returns STATUS_INVALID_DEVICE_REQUEST for a file on a FAT or CDFS file system.
- Windows Vista returns STATUS_INVALID_DEVICE_REQUEST for a file on a FAT or CDFS file system.
- Windows Server 2008 returns STATUS_INVALID_DEVICE_REQUEST for a file on a FAT or CDFS file system.
- Windows 7 returns STATUS_INVALID_DEVICE_REQUEST for a file on a FAT or CDFS file system.
- Windows Server 2008 R2 returns STATUS_INVALID_DEVICE_REQUEST for a file on a FAT or CDFS file system.

<30> Section 2.3.30: 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.

<31> Section 2.3.32: 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.

<32> Section 2.3.33: The FSCTL_GET_RETRIEVAL_POINTERS_AND_REFCOUNT request is supported only on ReFS and Windows 10 v1703 operating system and later and Windows Server 2019 and later.

<33> Section 2.3.34: On an ReFS volume, all alternate data streams are resident and all default data streams are non-resident. A resident data stream has no retrieval pointers to return.

<34> Section 2.3.36: Windows NT operating system, Windows 2000, Windows XP, Windows Server 2003, Windows Vista, Windows Server 2008, Windows 7, and Windows Server 2008 R2 operating system support the FSCTL_IS_PATHNAME_VALID Request (section 2.3.35) and return STATUS_SUCCESS whenever this request is invoked.

<35> Section 2.3.39: This operation is supported only by the NTFS and ReFS file systems.

<36> Section 2.3.41: Offload Read operations are supported only by the NTFS file system running on Windows 8 and later.

<37> Section 2.3.41: Clients and servers cannot depend on the **TokenTimeToLive** field as a true timer, because vendors can choose to ignore the requested TTL value or can implement the TTL counter in a vendor-specific manner. The **TokenTimeToLive** field can be interpreted as a hint.

<38> Section 2.3.41: The generated Token can represent less data than the requested amount; this information is contained in the **TransferLength** field in the FSCTL_OFFLOAD_READ_OUTPUT data element; for more information, see section 2.3.42.

<39> Section 2.3.42: In the following two cases, a well-known token, STORAGE_OFFLOAD_TOKEN_TYPE_ZERO_DATA, is returned, even if the target volume does not support Offload Read:

- If FSCTL_OFFLOAD_READ_INPUT.FileOffset is greater than or equal to the Valid Data Length (VDL) of the file.
- Or, if FSCTL_OFFLOAD_READ_INPUT.CopyLength is 0.

<40> Section 2.3.42: File reads can start beyond the Valid Data Length (VDL), but not beyond EOF.

<41> Section 2.3.43: Offload Write operations are supported only by the NTFS file system running on Windows 8 and later.

<42> Section 2.3.43: The FSCTL_OFFLOAD_READ and FSCTL_OFFLOAD_WRITE is used by Windows to copy large files.

When copying files, Windows avoids using offload operations on volumes that do not support offload. However, it is possible that the source volume and the destination volume both support offload, yet offload cannot occur from the source volume to the destination volume because of SAN topology or storage array compatibility issues. When this happens, Windows avoids repeated offload attempts between these two volumes.

There is currently no reliable way to detect unreachable volume pairs because there is no unique status code for this scenario. STATUS_INVALID_TOKEN can be returned for a variety of

reasons including unreachable volume pairs or a token expiration due to time-out.

In a best effort to detect unreachable volume pairs, Windows assumes a pair of volumes is not reachable if all the following are true:

- This is the first token write on the file stream.
- The FSCTL_OFFLOAD_WRITE request returns with a status code of STATUS_INVALID_TOKEN.
- The Offload Write operation is made at offset 0 in the destination file.

Windows chunks data for Offload Write operations into segments of 256 MB, a size that is subject to change.

<43> Section 2.3.44: While it is valid to issue a single Offload Write operation for the full contents of a file, the Win32 CopyFileEx API does not perform this. Instead, CopyFileEx issues Offload Write operations in 256-MB chunks so that components like Explorer can show proper progress of file copy operations.

<44> Section 2.3.52: Each entry in the output array contains an offset and a length that indicates a range in the file that can contain nonzero data. The actual nonzero data, if any, is somewhere within this range, and the calling application scans further within the range to locate it and determines if it really is valid data. Multiple instances of valid data can exist within the range.

<45> Section 2.3.52: Sparse files are supported by Windows 2000, Windows XP, Windows Server 2003, Windows Vista, Windows Server 2008, Windows 7, and Windows Server 2008 R2. The NTFS file system rounds down the input file offset to a 65,536-byte (64-kilobyte) boundary, rounds up the length to a convenient boundary, and then begins to walk through the file.

<46> Section 2.3.52: Windows does not track every piece of zero (0) or nonzero data. Because zero (0) is often perfectly legal data, it would be misleading. Instead, the system tracks ranges in which disk space is allocated. Where no disk space is allocated, all data bytes within that range for **Length** bytes from **FileOffset** are assumed to be zero (0) (when data is read, NTFS returns a zero for every byte in a sparse region). Allocated storage can contain zero (0) or nonzero data. So all that this operation does is return information on parts of the file where nonzero data might be located. It is up to the application to scan these parts of the file in accordance with the application's data conventions.

<47> Section 2.3.55: This region usage flag can only be specified for volumes using the NTFS file system.

<48> Section 2.3.55: This region usage flag can only be specified for volumes using the ReFS file system.

<49> Section 2.3.56.1: The NTFS file system is the only file system that returns this region usage value.

<50> Section 2.3.56.1: The ReFS file system is the only file system that returns this region usage value.

<51> Section 2.3.58: The following is the Windows UDF File System Support table. It lists the UDF revisions and "builds" (VAT/Spared/Write) that are supported by each covered version of Windows.

Expand table

Windows	UDF V1.02	UDF V1.5	UDF V2.01	UDF V2.5	UDF 2.6
95 / 95OSR2	-	-	-	-	-
Windows 98	Read	-	-	-	-
Windows NT	-	-	-	-	-
Windows 2000	Read	Read	-	-	-
Windows XP	Read	Read	Read	-	-
Windows Server 2003	Read	Read	Read	-	-
Windows Vista	Read/Write	Read/Write	Read/Write	Read/Write	-
Windows 7 and later and Windows Server 2008 and later	Read/Write	Read/Write	Read/Write	Read/Write	Read/Write

Note If Read of a given UDF version is supported, then reading of all UDF variants of that version are supported (VAT, Sparing and Simple). If Read/Write of a given UDF version is supported, then reading/writing of all UDF variants of that version are supported (VAT, Sparing and Simple).

<52> Section 2.3.58: The Windows UDF implementation pads the entire **CopyrightInfo** field with NULLs.

<53> Section 2.3.58: The Windows UDF implementation pads the entire **AbstractInfo** field with NULLs.

<54> Section 2.3.58: When the volume is formatted on Windows, this value is set to "*Microsoft Windows" followed by Unicode NULLs.

<55> Section 2.3.58: When the volume is written to on a Windows system, this value is set to "*Microsoft Windows" followed by Unicode NULLs.

<56> Section 2.3.61: This operation is supported by both the NTFS and ReFS file systems.

<57> Section 2.3.61: Currently supported values are 2 or 3. The **MinMajorVersion** is <= **MaxMajorVersion**.

<58> Section 2.3.61: Currently supported values are 2 or 3. The **MinMajorVersion** is <= **MaxMajorVersion**.

<59> Section 2.3.62.1: The major version number is 2 for file systems created on Windows 2000, Windows XP, Windows Server 2003, Windows Vista, Windows Server 2008, Windows 7, and Windows Server 2008 R2.

<60> Section 2.3.62.1: The minor version number is 0 for file systems created on Windows 2000, Windows XP, Windows Server 2003, Windows Vista, Windows Server 2008, Windows 7, and Windows Server 2008 R2.

<61> Section 2.3.62.2: The contents of a USN_RECORD_V2 or USN_RECORD_V3 element returned by this FSCTL is a partially populated record compared to the fully populated records returned by a local-only FSCTL FSCTL_READ_USN_JOURNAL.

<62> Section 2.3.67: Equivalent to COMPRESSION_FORMAT_LZNT1.

<63> Section 2.3.67: The LZNT1 is the only compression algorithm implemented on Windows 2000, Windows XP, Windows Server 2003, Windows Vista, Windows Server 2008, Windows 7, and Windows Server 2008 R2. Therefore, requests for COMPRESSION_FORMAT_DEFAULT and COMPRESSION_FORMAT_LZNT1 are equivalent from the server's perspective.

<64> Section 2.3.71: This message is implemented only on NTFS, and it is only for private use by the Encrypted File System (EFS). EFS issues this message locally on the machine that physically contains the file, notifying NTFS of a change in the file/stream attributes and causing NTFS to invoke the EFS callback that does the actual work of encrypting/decrypting streams.

This message is not used by any other component other than local EFS on Windows. It is not sent by the SMB1 and SMB2 client redirectors, nor is it accepted by an SMB2 server. In order to manipulate the encryption state of files and streams, clients use EFS and the EFSRPC protocol specified in [MS-EFSR].

<65> Section 2.3.71: The SMB1 server does not currently fail the FSCTL_SET_ENCRYPTION Request (section 2.3.71) if received. A QFE is planned to address this issue for the SMB1 server.

<66> Section 2.3.71: Windows sets the FILE_ATTRIBUTE_ENCRYPTED flag in the duplicate information file attributes field, and invokes the EFS callback which then creates the $EFS attribute.

<67> Section 2.3.71: Windows takes the following actions to clear encryption:

- Clears the FILE_ATTRIBUTE_ENCRYPTED flag in the duplicate information file attributes field.
- Invokes the EFS callback, which removes the $EFS attribute.

<68> Section 2.3.71: Windows takes the following actions to set encryption on a stream:

- If the stream is a resident user data stream, converts it to non-resident.
- Sets ATTRIBUTE_FLAG_ENCRYPTED in the attribute header.
- Invokes the EFS callback to generate an encryption context for this stream.

Note that if this is called during the creation of a named data attribute on a file with an empty unnamed data attribute, then the unnamed data attribute will be converted to non-resident and its attribute header flag will be set to encrypted.

Also note that this will set the FILE_ATTRIBUTE_ENCRYPTED flag if it is the first stream on the file that is encrypted.

<69> Section 2.3.71: Windows clears the ATTRIBUTE_FLAG_ENCRYPTED flag from the attribute header and invokes the EFS callback to free the encryption context for the stream.

<70> Section 2.3.71: The **Private** field is a placeholder marking the beginning of the private portion of the encryption buffer structure. This portion of the structure is meaningful only to EFS, because all the information necessary to fill (making a well-formed request) is private to EFS. Windows uses the EFSRPC protocol as specified in [MS-EFSR] to manipulate file encryption state.

<71> Section 2.3.72: An FSCTL_SET_ENCRYPTION operation never succeeds unless it is requested by the Encrypted File System (EFS), because the information necessary to make a well-formed request is visible only to EFS, as FSCTL_SET_ENCRYPTION is only for private use by EFS. Windows uses the EFSRPC protocol as specified in [MS-EFSR] to manipulate file encryption state.

<72> Section 2.3.72: On Windows, encryption requires NTFS major version 2 or greater.

<73> Section 2.3.72: Windows returns this error code if the NTFS encryption driver is not loaded or the FILE_CLEAR_ENCRYPTION operation was requested on a file containing a stream that is still marked as encrypted.

<74> Section 2.3.72: Windows returns this error code if the $INDEX_ROOT attribute of the directory that was trying to be encrypted, could not be found.

<75> Section 2.3.73: The FSCTL_SET_INTEGRITY_INFORMATION Request (section 2.3.73) message is supported only by the ReFS file system.

<76> Section 2.3.75: The FSCTL_SET_INTEGRITY_INFORMATION_EX Request message is supported only by Windows Server 2022 and later, and Windows 11, version 22H2 operating system and later. FSCTL_SET_INTEGRITY_INFORMATION_EX is processed as described on systems updated with [MSKB-5014019], [MSKB-5014021], [MSKB-5014022], [MSKB-5014023], or [MSKB-5014702].

<77> Section 2.3.77: 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.

<78> Section 2.3.77: All Windows versions: This request is never sent to a remote server.

<79> Section 2.3.79: 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.

<80> Section 2.3.83: This operation is supported by both the NTFS and ReFS file systems. ReFS supports this operation for conventional streams, but not for integrity streams, in Windows 8 and Windows Server 2012. ReFS supports this operation for both conventional and integrity streams in Windows 8.1 and later.

<81> Section 2.3.83: NTFS does not attempt to recover a failed unsparse operation by "resparsing".

<82> Section 2.3.83: Neither NFTS or ReFS deallocate existing clusters.

<83> Section 2.3.85: This operation is supported by both the NTFS and ReFS file systems.

Upon 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 zero (0) without extending the file size.

ReFS supports FSCTL_SET_ ZERO_DATA for conventional file streams, but not for integrity file streams, in Windows 8 and Windows Server 2012. ReFS supports FSCTL_SET_ ZERO_DATA for both conventional and integrity file streams in Windows 8.1 and later and Windows Server 2012 R2 operating system and later.

Upon receipt of this message, ReFS might deallocate disk space in the file if the file is stored on a ReFS volume and the file is sparse. 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 zero (0) without extending the file size.

<84> Section 2.3.87: This message is implemented only by NTFS, which is supported on Windows NT, Windows XP, Windows 2000, Windows Server 2003, Windows Vista, Windows Server 2008, Windows 7, and Windows Server 2008 R2.

<85> Section 2.3.89: Both the source and destination file names 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.

<86> Section 2.3.91: All Windows Server versions return STATUS_NOT_IMPLEMENTED.

<87> Section 2.4: The FileHardLinkInformation, FileIdGlobalTxDirectoryInformation, FileMailslotQueryInformation, FileMailslotSetInformation, FileNameInformation, FileObjectIdInformation, FileReparsePointInformation, FileSfioReserveInformation, FileStandardLinkInformation, and FileTrackingInformation file information classes are intended for local use only; the server will fail them with STATUS_NOT_SUPPORTED.

<88> 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.

<89> Section 2.4: FileDispositionInformationEx information class is supported in Windows 10 v1607 operating system and later and Windows Server 2016 and later.

<90> Section 2.4: FileId64ExtdBothDirectoryInformation information class is supported in the NTFS and ReFS file systems in Windows 11, version 23H2 operating system and later and Windows Server 2022, 23H2 operating system and later.

<91> Section 2.4: FileId64ExtdDirectoryInformation information class is supported in the NTFS and ReFS file systems in Windows 11, version 23H2 and later and Windows Server 2022, 23H2 and later.

<92> Section 2.4: FileIdAllExtdBothDirectoryInformation information class is supported in the NTFS and ReFS file systems in Windows 11, version 23H2 and later and Windows Server 2022, 23H2 and later.

<93> Section 2.4: FileIdAllExtdDirectoryInformation information class is supported in the NTFS and ReFS file systems in Windows 11, version 23H2 and later and Windows Server 2022, 23H2 and later.

<94> Section 2.4: The FileIdInformation information class is supported in the NTFS and ReFS file systems in Windows 8 and later and Windows Server 2012 and later.

<95> Section 2.4: This information class is not sent across the wire. In Windows, it is handled by the **IOManager** on the client. If this operation is sent to an SMB server, both SMB and SMB2 send the request to the **IOManager** on the server and perform normal processing of the operation.

<96> Section 2.4: Windows file systems do not implement this file information class; the server will fail it with STATUS_NOT_SUPPORTED.

<97> Section 2.4: Windows 10 v1803 and later and Windows Server v1803 and later allow remote FileNormalizedNameInformation query; other servers return STATUS_NOT_SUPPORTED.

<98> Section 2.4: The CIFS, SMB, and SMB2 protocols do not directly call this information class but use the structures associated with it.

<99> Section 2.4: FileRenameInformationEx information class is supported in Windows 10 v1607 and later and Windows Server 2016 and later.

<100> Section 2.4: Windows file systems do not implement this file information class; the server will fail it with STATUS_NOT_SUPPORTED.

<101> 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 is always 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 can 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).

<102> Section 2.4.4: NTFS rounds allocation size for resident files to a multiple of 8 bytes. When shrinking a resident file's allocation size using the FileAllocationInformation info class, the file remains resident with an allocation size rounded up to a multiple of 8 bytes. When extending a resident file's allocation size using the FileAllocationInformation info class, the file is converted to nonresident with an allocation size rounded up to a multiple of the cluster size.

<103> 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.

<104> 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. A driver or application can subsequently request that the file system resume updating 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 -2. The caller can set one, all, or any other combination of these three members to -1 and/or -2. Only the members that are set to -1 will be unaffected by I/O operations on the file handle; the other members will be updated as appropriate. This behavior is consistent across all file system types. Note that even though -1 and -2 can be used with the **CreationTime** field, they have no effect because file creation time is never updated in response to file system calls such as read and write.

Expand table

File system	Support value of -2
FAT	No
EXFAT	No
FAT32	No
Cdfs	No
UDFS	No
NTFS	Windows 8.1 and later, and Windows Server 2012 R2 and later
ReFS	Windows 10 v1507 operating system and later, and Windows Server 2016 and later

<105> Section 2.4.7: The file system updates the value of the **LastAccessTime** member 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. A driver or application can subsequently request that the file system resume updating 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 -2. The caller can set one, all, or any other combination of these three members to -1 and/or -2. Only the members that are set to -1 will be unaffected by I/O operations on the file handle; the other members will be updated as appropriate. This behavior is consistent across all file system types. Note that even though -1 and -2 can be used with the **CreationTime** field, they have no effect because file creation time is never updated in response to file system calls such as read and write.

Expand table

File system	Support value of -2
FAT	No

File system	Support value of -2
EXFAT	No
FAT32	No
Cdfs	No
UDFS	No
NTFS	Windows 8.1 and later, and Windows Server 2012 R2 and later
ReFS	Windows 10 v1507 and later, and Windows Server 2016 and later

<106> Section 2.4.7: The file system updates the value of the **LastWriteTime** member 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. A driver or application can subsequently request that the file system resume updating 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 -2. The caller can set one, all, or any other combination of these three members to -1 and/or -2. Only the members that are set to -1 will be unaffected by I/O operations on the file handle; the other members will be updated as appropriate. This behavior is consistent across all file system types. Note that even though -1 and -2 can be used with the **CreationTime** field, they have no effect because file creation time is never updated in response to file system calls such as read and write.

Expand table

File system	Support value of -2
FAT	No
EXFAT	No
FAT32	No
Cdfs	No
UDFS	No
NTFS	Windows 8.1 and later, and Windows Server 2012 R2 and later
ReFS	Windows 10 v1507 and later, and Windows Server 2016 and later

<107> Section 2.4.7: The file system updates the value of the **ChangeTime** member 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. A driver or application can subsequently request that the file system resume updating 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 -2. The caller can set one, all, or any other combination of these three members to -1 and/or -2. Only the members that are set to -1 will be unaffected by I/O operations on the file handle; the other members will be updated as appropriate. This behavior is consistent across all file system types. Note that even though -1 and -2 can be used with the **CreationTime** field, they have no effect because file creation time is never updated in response to file system calls such as read and write.

Expand table

File system	Support value of -2
FAT	No
EXFAT	No
FAT32	No
Cdfs	No
UDFS	No
NTFS	Windows 8.1 and later, and Windows Server 2012 R2 and later
ReFS	Windows 10 v1507 and later, and Windows Server 2016 and later

<108> Section 2.4.8: When using ReFS or NTFS, the position of a file within the parent directory is not fixed and can be changed at any time. Windows sets this value to zero for files on ReFS and NTFS file systems.

<109> Section 2.4.9: Windows 2000, Windows XP, Windows Server 2003, Windows Vista, Windows Server 2008, Windows 7, and Windows Server 2008 R2 implement only one compression algorithm, LZNT1. For more information, see [UASDC].

<110> 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 can vary. The default compression unit size based on this calculation is 64 kilobytes.

<111> Section 2.4.9: NTFS uses a value of 12 for the ChunkShift so that compression chunks are 4 kilobytes in size.

<112> Section 2.4.9: The value of this field depends on the cluster size set for the file system at initialization. NTFS uses a value of 12 by default because the default NTFS cluster size is 4 kilobytes. If an NTFS file system is initialized with a different cluster size, the value of **ClusterShift** would be log 2 of the cluster size for that file system.

<113> Section 2.4.10: When using ReFS or NTFS, the position of a file within the parent directory is not fixed and can be changed at any time. Windows sets this value to zero for files on ReFS and NTFS file systems.

<114> 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.

<115> Section 2.4.15: When using ReFS or NTFS, the position of a file within the parent directory is not fixed and can be changed at any time. Windows sets this value to zero for files on ReFS and NTFS file systems.

<116> Section 2.4.17: In Windows, both the NTFS and UDFS file systems support hard links. UDFS support of hard links was added in Windows Vista and Windows Server 2008.

<117> Section 2.4.18: When using ReFS or NTFS, the position of a file within the parent directory is not fixed and can be changed at any time. Windows sets this value to zero for files on ReFS and NTFS file systems.

<118> Section 2.4.18: The NTFS, ReFS, FAT, and exFAT file systems return a **FileId** value of 0 for the entry named ".." in directory query operations.

<119> Section 2.4.19: When using ReFS or NTFS, the position of a file within the parent directory is not fixed and can be changed at any time. Windows sets this value to zero for files on ReFS and NTFS file systems.

<120> Section 2.4.19: The NTFS, ReFS, FAT, and exFAT file systems return a **FileId** value of 0 for the entry named ".." in directory query operations.

<121> Section 2.4.20: When using ReFS or NTFS, the position of a file within the parent directory is not fixed and can be changed at any time. Windows sets this value to zero for files on ReFS and NTFS file systems.

<122> Section 2.4.20: The NTFS, ReFS, FAT, and exFAT file systems return a **FileId** value of 0 for the entry named ".." in directory query operations.

<123> Section 2.4.21: When using ReFS or NTFS, the position of a file within the parent directory is not fixed and can be changed at any time. Windows sets this value to zero for files on ReFS and NTFS file systems.

<124> Section 2.4.21: The NTFS, ReFS, FAT, and exFAT file systems return a **FileId** value of 0 for the entry named ".." in directory query operations.

<125> Section 2.4.22: When using ReFS or NTFS, the position of a file within the parent directory is not fixed and can be changed at any time. Windows sets this value to zero for files on ReFS and NTFS file systems.

<126> Section 2.4.22: The NTFS, ReFS, FAT, and exFAT file systems return a **FileId** value of 0 for the entry named ".." in directory query operations.

<127> Section 2.4.23: When using ReFS or NTFS, the position of a file within the parent directory is not fixed and can be changed at any time. Windows sets this value to zero for files on ReFS and NTFS file systems.

<128> Section 2.4.24: Windows-based SMB Version 1 servers set the **NextEntryOffset** field to the size of the current **FileIdFullDirectoryInformation** entry in bytes, if no other entries follow this one.

<129> Section 2.4.24: When using ReFS or NTFS, the position of a file within the parent directory is not fixed and can be changed at any time. Windows sets this value to zero for files on ReFS and NTFS file systems.

<130> Section 2.4.24: The NTFS, ReFS, FAT, and exFAT file systems return a **FileId** value of 0 for the entry named ".." in directory query operations.

<131> Section 2.4.25: When using ReFS or NTFS, the position of a file within the parent directory is not fixed and can be changed at any time. Windows sets this value to zero for files on ReFS and NTFS file systems.

<132> Section 2.4.25: The NTFS, ReFS, FAT, and exFAT file systems return a **FileId** value of 0 for the entry named ".." in directory query operations.

<133> Section 2.4.27: The NTFS, ReFS, FAT, and exFAT file systems return a **FileId** value of 0 for the entry named ".." in directory query operations.

<134> Section 2.4.28: In Windows, both the NTFS and UDFS file systems support hard links. UDFS support of hard links was added in Windows Vista and Windows Server 2008.

<135> Section 2.4.31: The **FileModeInformation** information class is not sent across the wire. In Windows, it is handled by the **IOManager** on the client. If this operation is sent to an SMB server, both SMB and SMB2 send the request to the **IOManager** on the server and perform normal processing of the operation.

<136> Section 2.4.31: This flag is cleared by the respective server application while processing the set operation in the following situations:

- SMB server on all supported versions of Windows if the file is not opened with a **DesiredAccess** field value that has the FILE_WRITE_DATA or FILE_APPEND_DATA bit set (see [MS-CIFS] section 2.2.4.64.1).
- SMB2 server on Windows Vista and Windows Server 2008 always.
- SMB2 server on Windows 7 and Windows Server 2008 R2 if the file is opened with a **CreateOptions** field value that has the FILE_NO_INTERMEDIATE_BUFFERING bit set (see [MS-SMB2] section 2.2.13).

<137> Section 2.4.33: When using ReFS or NTFS, the position of a file within the parent directory is not fixed and can be changed at any time. Windows sets this value to zero for files on ReFS and NTFS file systems.

<138> Section 2.4.34: This operation works on both remote and local handles.

<139> Section 2.4.35: This information class is implemented on ReFS and NTFS file systems. Other file systems return STATUS_INVALID_DEVICE_REQUEST.

<140> Section 2.4.36: The Microsoft ReFS, FAT, EXFAT, UDFS, and CDFS file systems do not support the use of **ObjectId**s and return a status code of STATUS_INVALID_DEVICE_REQUEST.

<141> Section 2.4.36: The Microsoft Distributed Link Tracking protocols (see [MS-DLTW] section 3.1.6) use the first type of object ID structure for link tracking.

<142> Section 2.4.36.1: 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.

<143> Section 2.4.40: Both the query and set FilePositionInformation operations are processed on the local client; therefore, these operations are not transmitted across the wire. The fact that these operations are processed on the client instead of the server is intended to be transparent to the client's usage of these operations.

If a server receives a request to set FilePositionInformation, the specified file position will be set on the remote handle, but its value will be ignored by future read/write operations. If a server receives a request to query FilePositionInformation, an undetermined value will be returned. For more information on how the **CurrentByteOffset** field is updated, see the [MS-FSA] sections for read and write operations.

<144> Section 2.4.40: Each read and write operation via the Server Message Block (SMB) Protocol [MS-SMB] and Server Message Block (SMB) Version 2 [MS-SMB2] protocols always provides an explicit starting offset, and thus is unaffected by the file position. Windows does not update the file position when read and write operations are performed via these protocols.

<145> Section 2.4.41: Query and set operations are supported only by the NTFS file system and are valid only on handles opened to the NTFS metadata file "\$Extend\$Quota:$Q:$INDEX_ALLOCATION".

<146> Section 2.4.43: FILE_RENAME_SUPPRESS_PIN_STATE_INHERITANCE is supported in Windows 10 v1709 operating system and later and Windows Server 2019 and later.

<147> Section 2.4.43: FILE_RENAME_SUPPRESS_STORAGE_RESERVE_INHERITANCE is supported in Windows 10 v1809 operating system and later and Windows Server 2019 and later.

<148> Section 2.4.43: FILE_RENAME_NO_INCREASE_AVAILABLE_SPACE is supported in Windows 10 v1809 and later and Windows Server 2019 and later.

<149> Section 2.4.43: FILE_RENAME_NO_DECREASE_AVAILABLE_SPACE is supported in Windows 10 v1809 and later and Windows Server 2019 and later.

<150> Section 2.4.43: FILE_RENAME_PRESERVE_AVAILABLE_SPACE is supported in Windows 10 v1809 and later and Windows Server 2019 and later.

<151> Section 2.4.43: FILE_RENAME_IGNORE_READONLY_ATTRIBUTE is supported in Windows 10 v1809 and later and Windows Server 2019 and later.

<152> Section 2.4.43: FILE_RENAME_FORCE_RESIZE_TARGET_SR is supported in Windows 10 v1903 operating system and later and Windows Server v1909 operating system and later.

<153> Section 2.4.43: FILE_RENAME_FORCE_RESIZE_SOURCE_SR is supported in Windows 10 v1903 and later and Windows Server v1909 and later.

<154> Section 2.4.43: FILE_RENAME_FORCE_RESIZE_SR is supported in Windows 10 v1903 and later and Windows Server v1909 and later.

<155> Section 2.4.46: In Windows 7 and Windows Server 2008 R2, the existing short name is deleted if the **FileNameLength** field in **FILE_NAME_INFORMATION** is zero. Previous Windows implementations return STATUS_INVALID_PARAMETER when the **FileNameLength** field is zero.

<156> Section 2.4.48: This information class is supported on Windows 7 and later and Windows Server 2008 R2 and later.

<157> Section 2.4.50: Windows supports the FileValidDataLengthInformation (section 2.4.50) information class in the ReFS, NTFS, FAT, FAT32, and EXFAT file systems.

<158> 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.

<159> Section 2.5: This file system information class is intended for local use only; the server will fail it with status STATUS_NOT_SUPPORTED.

<160> Section 2.5: This file system information class is intended for local use only; the server will fail it with status STATUS_NOT_SUPPORTED. Furthermore, this file information class is not implemented by any Windows file systems.

<161> Section 2.5: This file system information class is intended for local use only; the server will fail a "query" with STATUS_ACCESS_NOT_SUPPORTED, and the server will fail a "set" with STATUS_ACCESS_DENIED. Furthermore, this file information class is not implemented by any Windows file systems.

<162> Section 2.5.1: The FILE_SUPPORTS_USN_JOURNAL, FILE_SUPPORTS_OPEN_BY_FILE_ID, FILE_SUPPORTS_EXTENDED_ATTRIBUTES, and FILE_SUPPORTS_HARD_LINKS attributes are only available on Windows 7 and Windows Server 2008 R2.

The FILE_READ_ONLY_VOLUME attribute is only available on Windows XP, Windows Server 2003, Windows Vista, Windows Server 2008, Windows 7, and Windows Server 2008 R2.

The FILE_SUPPORT_INTEGRITY_STREAMS attribute is available only on ReFS/Windows 8.

<163> Section 2.5.1: Windows Vista, Windows Server 2008, Windows 7, and Windows Server 2008 R2 set this flag if the volume is formatted for NTFS 3.0 or higher.

<164> Section 2.5.1: Windows support for a volume formatted to NTFS version 3.0 or 3.1 is required for EFS use. NTFS versions 3.0 and 3.1 are supported on Windows 2000 and later. Support for FAT and EXFAT was added in Windows 10 v1607 and Windows Server 2016 and later.

<165> Section 2.5.1: NTFS file systems in Windows 10 v1607 and later and Windows Server 2016 and later support this flag.

<166> Section 2.5.1: NTFS file systems in Windows 10 v1607 and later and Windows Server 2016 and later support this flag.

<167> Section 2.5.1: Remote storage is provided by the Remote Storage service to create virtual disk storage from a tape or other storage media.

<168> Section 2.5.1: For the Microsoft ReFS, NTFS, FAT, and EXFAT file systems, this value is 255. For the Microsoft UDFS file system, this value is 254. For the Microsoft CDFS file system, this value is 110 for Joliet format and 221 otherwise.

<169> Section 2.5.1: Valid values for this field depend on the version of Windows that the server is running.

Expand table

Windows version	FAT	FAT16	FAT32	exFAT	NTFS	CDFS	UDF	CSVFS
Windows 8 and later and Windows Server 2012 operating system and later	X	X	X	X	X	X	X	X
Windows 7, Windows Server 2008 R2	X	X	X	X	X	X	X	
Windows Vista operating system with Service Pack 1 (SP1), Windows Server 2008, Windows Server 2008 R2	X	X	X	X	X	X	X	
Windows Vista RTM	X	X	X		X	X	X	
Windows XP	X	X	X		X	X		

<170> Section 2.5.2: Query and set operations are supported only by the NTFS file system, and the quota index information is saved in the NTFS metadata file "\$Extend\$Quota:$Q:$INDEX_ALLOCATION".

<171> Section 2.5.2: Logging makes an entry in the Windows application event log.

<172> Section 2.5.4: In Windows 2000, Windows XP, Windows Server 2003, Windows Vista, Windows Server 2008, Windows 7, and Windows Server 2008 R2, if per-user quotas are in use, this value can 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.

<173> Section 2.5.4: In Windows 2000, Windows XP, Windows Server 2003, Windows Vista, Windows Server 2008, Windows 7, and Windows Server 2008 R2, if per-user quotas are in use, this value can be less than the total number of free allocation units on the disk.

<174> Section 2.5.5: A maximum length of 32 characters is imposed for any Windows file system, though some file systems can impose a stricter limit. The Microsoft FAT file system supports volume labels that are 0 to 11 characters in length. ReFS and NTFS support volume labels that are 0 to 32 characters in length. All Unicode characters are permitted in a volume label with the exception of the NULL character, which is reserved for use as a string terminator.

<175> Section 2.5.6: The Microsoft ReFS, FAT, EXFAT, UDFS, and CDFS file systems do not support the use of object IDs and return a status code of STATUS_INVALID_PARAMETER.

<176> Section 2.5.6: Windows does not write information into the **ExtendedInfo** field for file systems.

<177> Section 2.5.7: This information class is only available in the following:

- Windows 8 and later
- Microsoft-implemented file systems including NTFS, ReFS, FAT, ExFAT, UDFS, and CDFS

<178> Section 2.5.7: This is also the reported physical sector size of the device for atomicity. Note that NTFS does basic sanitation to ensure this value does not cause unexpected application behavior. NTFS performs the following basic sanitization:

- Ensures that the reported physical sector size is greater than or equal to the logical sector size. If it is not, the value of this field is set to the logical sector size.
- Ensures that the reported physical sector size is a power of two. If it is not, the value of this field is set to the logical sector size.

<179> Section 2.5.7: This is the reported physical sector size of the device for performance. Note that NTFS does basic sanitation to ensure that this value does not cause unexpected application behavior. NTFS performs the following basic sanitization:

- Ensures that the reported physical sector size is greater than or equal to the logical sector size. If it is not, the value of this field is set to the logical sector size.
- Ensures that the reported physical sector size is a power of two. If it is not, the value of this field is set to the logical sector size.

<180> Section 2.5.7: A client can interpret this field as the unit for which NTFS guarantees an atomic operation. NTFS calculates the value of this field as follows:

- Retrieve the physical sector size the device reports for atomicity, and store in *x*.
- Validate that the value *x* is greater than or equal to the logical sector size. If it is not, set *x* to the logical sector size.
- Validate that the value *x* is a power of two. If it is not, set *x* to the logical sector size.
- Validate that the value *x* is less than or equal to the system page size defined in [MS-FSA] section 2.1.1.1. If it is not, set *x* to the system page size defined in [MS-FSA] section 2.1.1.1.

<181> Section 2.5.7: In this example, a storage device has a logical sector of 512 bytes, a physical sector of 4 KB (with eight logical sectors in a physical sector), and an offset of three

logical sectors. The **ByteOffsetForSectorAlignment** field is therefore calculated as 3 * LogicalBytesPerSector = 1536 bytes.

Expand table

LBA	#	#	#	0	1	2	3	4	5	6	7	8	9	10	1	2	3	4	5	6	7	8	9	20
Physical Sector	0								1								2							

<182> Section 2.5.8: In Windows 2000, Windows XP, Windows Server 2003, Windows Vista, Windows Server 2008, Windows 7, and Windows Server 2008 R2, if per-user quotas are in use, this value can 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.

<183> Section 2.5.8: In Windows 2000, Windows XP, Windows Server 2003, Windows Vista, Windows Server 2008, Windows 7, and Windows Server 2008 R2, if per-user quotas are in use, this value can be less than the total number of free allocation units on the disk.

<184> Section 2.5.9: A maximum length of 32 characters is imposed for any Windows file system, though some file systems can impose a stricter limit. The Microsoft FAT file system supports volume labels that are 0 to 11 characters in length. NTFS supports volume labels that are 0 to 32 characters in length. All Unicode characters are permitted in a volume label with the exception of the NULL character, which is reserved for use as a string terminator.

<185> Section 2.5.9: This value is TRUE for NTFS and FALSE for other file systems implemented by Windows.

<186> Section 2.5.10: A driver can skip the full check for appcontainers by setting this characteristic on its device object.

<187> Section 2.6: The Windows file system does not persist the FILE_ATTRIBUTE_NORMAL flag. When getting attributes via the FileAttributeTagInformation (section 2.4.6) information class, a client will receive the FILE_ATTRIBUTE_NORMAL flag only if no other attributes were set. Some examples: If a client sets the attributes as [FILE_ATTRIBUTE_HIDDEN | FILE_ATTRIBUTE_NORMAL], the client will see just [FILE_ATTRIBUTE_HIDDEN] when it gets the attributes. If the client sets the attributes as [FILE_ATTRIBUTE_NORMAL], the client will see [FILE_ATTRIBUTE_NORMAL] when it gets the attributes.

<188> Section 2.6: Only ReFS supports this attribute.

<189> Section 2.6: Only NTFS and ReFS support this attribute.

<190> Section 2.6: Only NTFS and ReFS support this attribute.

<191> Section 2.6: Only NTFS and ReFS support this attribute.

<192> Section 2.6: Only NTFS and ReFS support this attribute.

<193> Section 2.6: Only NTFS and ReFS support this attribute.

<194> Section 2.7.1: For FILE_ACTION_REMOVED_BY_DELETE, FILE_ACTION_ID_NOT_TUNNELLED, and FILE_ACTION_TUNNELLED_ID_COLLISION only NTFS supports the special directory "\$Extend\$ObjId:$O:$INDEX_ALLOCATION".

7 Change Tracking

This section identifies changes that were made to this document since the last release. Changes are classified as Major, Minor, or None.

The revision class **Major** means that the technical content in the document was significantly revised. Major changes affect protocol interoperability or implementation. Examples of major changes are:

- A document revision that incorporates changes to interoperability requirements.
- A document revision that captures changes to protocol functionality.

The revision class **Minor** means that the meaning of the technical content was clarified. Minor changes do not affect protocol interoperability or implementation. Examples of minor changes are updates to clarify ambiguity at the sentence, paragraph, or table level.

The revision class **None** means that no new technical changes were introduced. Minor editorial and formatting changes may have been made, but the relevant technical content is identical to the last released version.

The changes made to this document are listed in the following table. For more information, please contact dochelp@microsoft.com.

Expand table

Section	Description	Revision class
2.4 File Information Classes	30546 : Updated the file information class "FileDispositionInformationEx" from Local to Remote.	Major
2.4 File Information Classes	30547 : Updated the file information class "FileRenameInformationEx" from Local to Remote.	Major
2.4.11 FileDispositionInformation	30420 : Updated information for FileDispositionInformation.	Major
2.4.12 FileDispositionInformationEx	30420 : Updated information for FileDispositionInformationEx.	Major

Last updated on 11/21/2025

8 Index

Article04/07/2025

FILE_NAME_INFORMATION packet

FILE_NOTIFY_INFORMATION packet

FILE_OBJECTID_BUFFER data element

FILE_OBJECTID_BUFFER_Type_1 packet

FILE_OBJECTID_BUFFER_Type_2 packet

FILE_OBJECTID_INFORMATION_TYPE_1 packet

FILE_OBJECTID_INFORMATION_TYPE_2 packet

FILE_POSITION_INFORMATION packet

FILE_QUOTA_INFORMATION packet

FILE_REGION_INFO packet

FILE_RENAME_INFORMATION_TYPE_1 packet

FILE_RENAME_INFORMATION_TYPE_2 packet

FILE_SET_DEFECT_MGMT_BUFFER packet

FileAllInformation packet

FileAllocationInformation

FileAlternateNameInformation information class

FileAttributeTagInformation packet

FileBasicInformation packet

FileBothDirectoryInformation packet

FileCompressionInformation packet

FileDirectoryInformation packet

FileDispositionInformation packet

FileEaInformation packet

FileEndOfFileInformation packet

FileFsAttributeInformation packet

FileFsControlInformation packet

FileFsDeviceInformation packet

FileFsDriverPathInformation packet

FileFsFullSizeInformation packet

FileFsLabelInformation packet

FileFsObjectIdInformation packet

FileFsSectorSizeInformation packet

FileFsSizeInformation packet

FileFsVolumeInformation packet

FileFullDirectoryInformation packet

FileFullEaInformation packet

FileHardLinkInformation packet

FileIdBothDirectoryInformation packet

FileIdFullDirectoryInformation packet

FileIdGlobalTxDirectoryInformation packet

FileInternalInformation packet

FileLinkInformation packet (section 2.4.28, section 2.4.28.1, section 2.4.28.2)

FileMailslotQueryInformation packet

FileMailslotSetInformation packet

FileNameInformation information class

FileNamesInformation packet

FileNetworkOpenInformation packet

FileObjectIdInformation information class

FilePipeInformation packet

FilePipeLocalInformation packet

FSCTL_GET_NTFS_VOLUME_DATA request

FSCTL_GET_OBJECT_ID reply

FSCTL_GET_OBJECT_ID request

FSCTL_GET_REFS_VOLUME_DATA reply

FSCTL_GET_REFS_VOLUME_DATA request

FSCTL_GET_REFS_VOLUME_DATA_Reply packet

FSCTL_GET_REPARSE_POINT reply

FSCTL_GET_REPARSE_POINT request

FSCTL_GET_RETRIEVAL_POINTERS_Reply packet

FSCTL_GET_RETRIEVAL_POINTERS_Request packet

FSCTL_IS_PATHNAME_VALID reply

FSCTL_IS_PATHNAME_VALID_Request packet

FSCTL_LMR_SET_LINK_TRACKING_INFORMATION reply

FSCTL_LMR_SET_LINK_TRACKING_INFORMATION request

FSCTL_OFFLOAD_READ_INPUT packet

FSCTL_OFFLOAD_READ_OUTPUT packet

FSCTL_OFFLOAD_WRITE_INPUT packet

FSCTL_OFFLOAD_WRITE_OUTPUT packet

FSCTL_PIPE_PEEK packet

FSCTL_PIPE_PEEK reply

FSCTL_PIPE_PEEK request

FSCTL_PIPE_TRANSCEIVE reply

FSCTL_PIPE_TRANSCEIVE request

FSCTL_PIPE_WAIT reply

FSCTL_PIPE_WAIT_Request packet