D

CIFS Technical Reference

> The only spec I trust
> is written in C.
>
> — Andrew Bartlett,
> Samba Team

The SNIA CIFS Technical Reference is included by kind permission of the Storage Network Industry Association (SNIA). The author would like to thank the SNIA, particularly the members of the SNIA CIFS Technical Work Group, for making this document available. Please see page v of the CIFS Technical Reference for SNIA copyright information.

Please Note: The SNIA CIFS Technical Reference does not fall under the same copyright and licensing terms as the rest of the book. Special permission was obtained in order to include the CIFS Technical Reference in this book.

SNIA
Storage Networking Industry Association

Common Internet File System (CIFS) Technical Reference

Revision: 1.0

"Publication of this *SNIA Technical Proposal* has been approved by the SNIA. This document represents a stable proposal for use as agreed upon by the SNIA CIFS Technical Work Group. The SNIA does not endorse this proposal for any other purpose than the use described. This proposal may not represent the preferred mode, and the SNIA may update, replace, or release competing proposal at any time. If the intended audience for this release is a liaison standards body, the future support and revision of this proposal may be outside the control of the SNIA or originating SNIA CIFS Technical Work Group. Suggestion for revision should be directed to snia-cifs@snia.org"

SNIA Technical Proposal

USE OF THIS DOCUMENT IS GOVERNED BY THE TERMS AND CONDITIONS SPECIFIED ON PAGES iii-v

Release Date: 3/1/2002

Revision History

Date	By:	Comments
Feb 27, 2002	SNIA CIFS Technical Work Group	Version 1.0

Suggestion for changes or modifications to this document should be sent to the SNIA CIFS Technical Work Group at snia-cifs@snia.org

Abstract

The Common Internet File System (CIFS) is a file sharing protocol. Client systems use this protocol to request file access services from server systems over a network. It is based on the Server Message Block protocol widely in use by personal computers and workstations running a wide variety of operating systems. This document is a collaborative effort to produce more comprehensive documentation of the network protocol used by existing CIFS (Common Internet File System) implementations. Based on the widely used SMB (Server Message Block) network protocol, CIFS has become a key file sharing protocol due to its widespread distribution and its inclusion of enhancements that improve its suitability for internet authoring and file sharing. It is an integral part of workstation and server operating systems as well as embedded and appliance systems. In addition there has been a recent expansion of NAS (Network Attached Storage) and SAN-like (Storage Area Network) network storage server products based on CIFS. Although primarily a file sharing and authoring protocol, CIFS assumes even more importance due to the indirect use of CIFS as a transport protocol for various higher level NT and Windows9x communication protocols, as well as for network printing, resource location services, remote management/administration, network authentication (secure establishment services) and RPC (Remote Procedure Calls).

Intended Usage

The improved CIFS documentation, used as a development aid, will assist in decreased time-to-market for product developers and improved interoperability for products in the market place. It is the intent of the SNIA that this document reflect the best information available about the CIFS protocol. In certain places within the document indicated by MISSING, additional information is needed. The CIFS Technical Reference will be maintained by SNIA with the assistance of the collaborating organizations. This is not a standards document nor CIFS specification. It is a best effort at documenting the CIFS protocol as used by existing implementations. Inaccuracies or errors can be brought to the attention of the SNIA as well as new information on the existing protocol or new implementations. As new information or new implementations become available, it is the desire of the SNIA to collect and evaluate this information for possible incorporation into any future CIFS documentation that the SNIA CIFS documentation work group may choose to create.

DISCLAIMER OF WARRANTIES AND REPRESENTATIONS

This document is provided “as is”, without any express or implied warranties or representations of any kind. Without limitation, there is no warranty of merchantability, no warranty of noninfringement, and no warranty of fitness for a particular purpose. All such warranties are expressly disclaimed.

The SNIA and the SNIA member organizations do not warrant or assume any responsibility for the accuracy or completeness of any information, text, graphics, links, cross-references, or other items contained herein.

No express or implied license to any intellectual property exists due to the presentation, publication, distribution, or other dissemination of this document, or due to any use or implementation based on the subject matter in this document.

This document is an informal Technical Reference and not a formal Standards Document or formal specification intended for adoption as a Standard. By releasing this document, the SNIA and the SNIA member organizations are neither guaranteeing nor implying that any CIFS implementation(s) distributed or sold by them, presently or in the future, are compliant or compatible with the implementation(s) described in this document. The release of this document does not prevent SNIA or any SNIA member organization from modifying and/or extending their CIFS implementation(s) at any time.

LIMITATION OF LIABILITY

The SNIA and the SNIA member organizations are not liable for any damages whatsoever arising out of the use of or inability to use this document, even if the SNIA or any SNIA member organization has been notified of the possibility of such damages.

INTELLECTUAL PROPERTY RIGHTS

The SNIA and the SNIA member organizations take no position regarding the validity or scope of any intellectual property or other rights that might be claimed to pertain to the implementation or use of the technology described in this document or the extent to which any license under such rights might or might not be available; neither do they represent that they have made any effort to identify any such rights.

COPYRIGHT AND USAGE AGREEMENT

The SNIA hereby grants permission for individuals to use this document for personal use only, and for corporations and other business entities to use this document for internal use only (including internal copying, distribution, and display) provided that:

1. Any text, diagram, chart, table or definition reproduced must be reproduced in its entirety with no alteration,
2. No modification or creation of derivative documents based on this document, or any part of this document, is allowed, and
3. Any document, printed or electronic, in which material from this document (or any portion hereof) is reproduced must acknowledge the SNIA copyright on that material, and must credit the SNIA for granting permission for its reuse.

Other than as explicitly provided above, you may not make any commercial use of this document, sell any or all of this document, or distribute this document to third parties. All rights not explicitly granted are expressly reserved to SNIA.

Permission to use this document for purposes other than those enumerated above may be requested by e-mailing snia-tc@snia.org; please include the identity of the requesting individual and/or company and a brief description of the purpose, nature, and scope of the requested use.

Acknowledgements

The SNIA CIFS Documentation is a cooperative effort of the SNIA CIFS Documentation Work Group, bringing together the perspectives of system architects and developers from diverse backgrounds and perspectives in the storage industry. An effort of this scope could only be successful with support from each of the SNIA member organizations that sponsored the individuals contributing their time and knowledge to the creation and review of this document. The SNIA Board of Directors would like to extend its gratitude to this dedicated group of individuals and their sponsoring companies:

Work Group Chairman	Jim Norton, IBM
Co-Author	Bob Mastors, EMC
Co-Author	Byron Deadwiler, Hewlett-Packard
Co-Author	Bob Griswold & Jason Goodman, Microsoft
Co-Author	Christopher R. Hertel, Univ. of Minnesota
Co-Author	Dennis Chapman, Network Appliance
Co-Author	George Colley, Thursby Software Systems
Co-Author	Steve French, IBM
Co-Author	Tamir Ram, Veritas
The companies of the SNIA CIFS Documentation	Work Group reflector: ADIC, AMI, Cereva, CommVault, EMC, Eurologic, HP, IBM, KOM Networks, LSI Logic, Microsoft, Network Appliance, Novell, NSS, Quantum, Samba and Veritas

Table of Contents

Common Internet File System (CIFS)

1. Introduction

This document describes the file sharing protocol for a Common Internet File System (CIFS). CIFS is intended to provide an open cross-platform mechanism for client systems to request file services from server systems over a network. It is based on the standard Server Message Block (SMB) protocol widely in use by personal computers and workstations running a wide variety of operating systems. An earlier version of this protocol was documented as part of the X/OPEN (now Open Group) CAE series of standards [7]; this document updates the document to include the latest shipping versions, and is published to allow the creation of implementations that inter-operate with those implementations.

The scope of this document is limited to describing requests and responses for file services. Separate documents exist for clients requesting services other than file services, e.g. print services.

Use of the Internet and the World Wide Web has been characterized by read-only access. Existing protocols such as FTP are good solutions for one-way file transfer. However, new read/write interfaces will become increasingly necessary as the Internet becomes more interactive and collaborative. Adoption of a common file sharing protocol having modern semantics such as shared files, byte-range locking, coherent caching, change notification, replicated storage, etc. would provide important benefits to the Internet community.

1.1. Summary of features

The protocol supports the following features:

- File access
- File and record locking
- Safe caching, read-ahead, and write-behind
- File change notification
- Protocol version negotiation
- Extended attributes
- Distributed replicated virtual volumes
- Server name resolution independence
- Batched requests
- Unicode file names

1.1.1. File access

The protocol supports the usual set of file operations: open, close, read, write, and seek.

1.1.2. File and record locking

The protocol supports file and record locking, as well as unlocked access to files. Applications that lock files cannot be improperly interfered with by applications that do not; once a file or record is locked, non-locking applications are denied access to the file.

1.1.3. Safe caching, read-ahead, and write-behind

The protocol supports caching, read-ahead, and write-behind, even for unlocked files, as long as they are safe. All these optimizations are safe as long as only one client is accessing a file; read-caching and read-ahead are safe with many clients accessing a file as long as all are just reading. If many clients are writing a file simultaneously, then none are safe, and all file operations have to go to the server. The protocol notifies all clients accessing a file of changes in the number and access mode of clients accessing the file, so that they can use the most optimized safe access method.

1.1.4. File change notification

Applications can register with a server to be notified if and when file or directory contents are modified. They can use this to (for example) know when a display needs to be refreshed, without having to constantly poll the server.

1.1.5. Protocol version negotiation

There are several different versions and sub-versions of this protocol; a particular version is referred to as a dialect. When two machines first come into network contact they negotiate the dialect to be used. Different dialects can include both new messages as well as changes to the fields and semantics of existing messages in other dialects.

1.1.6. Extended attributes

In addition to many built-in file attributes, such as creation and modification times, non-file system attributes can be added by applications, such as the author's name, content description, etc.

1.1.7. Distributed replicated virtual volumes

The protocol supports file system subtrees which look like to clients as if they are on a single volume and server, but which actually span multiple volumes and servers. The files and directories of such a subtree can be physically moved to different servers, and their names do not have to change, isolating clients from changes in the server configuration. These subtrees can also be transparently replicated for load sharing and fault tolerance. When a client requests a file, the protocol uses referrals to transparently direct a client to the server that stores it.

1.1.8. Server name resolution independence

The protocol allows clients to resolve server names using any name resolution mechanism. In particular, it allows using the DNS, permitting access to the file systems of other organizations over the Internet, or hierarchical organization of servers' names within an organization. Earlier versions of the protocol only supported a flat server name space.

1.1.9. Batched requests

The protocol supports the batching of multiple requests into a single message, in order to minimize round trip latencies, even when a later request depends on the results of an earlier one.

1.1.10. Obsolescence

Throughout this document, references are made to obsolescent elements of the CIFS protocol. Note that these obsolescent elements are still observed in implementations. The “obsolescent” label only describes that these elements may be removed from implementations, in the future.

2. Protocol Operation Overview

In order to access a file on a server, a client has to:

- Parse the full file name to determine the server name, and the relative name within that server
- Resolve the server name to a transport address (this may be cached)
- Make a connection to the server (if no connection is already available)
- Exchange CIFS messages (see below for an example)

This process may be repeated as many times as desired. Once the connection has been idle for a while, it may be torn down.

2.1. Server Name Determination

How the client determines the name of the server and the relative name within the server is outside of the scope of this document. However, just for expository purposes, here are three examples.

In the URL "file://fs.megacorp.com/users/fred/stuff.txt", the client could take the part between the leading double slashes and the next slash as the server name and the remainder as the relative name – in this example "fs.megacorp.com" and "/users/fred/stuff.txt", respectively.

In the path name "\\corpserver\public\policy.doc" the client could take the part between the leading double backslashes and the next slash as the server name, and the remainder as the relative name -- in this example, "corpserver" and "\public\policy.doc" respectively.

In the path name "x:\policy.doc" the client could use "x" as an index into a table that contains a server name and a file name prefix. If the contents of such a table for "x" were "corpserver" and "\public", then the server name and relative name would be the same as in the previous example.

2.2. Server Name Resolution

Like server name determination, how the client resolves the name to the transport address of the server is outside the scope of this document. All that is required by CIFS is that a CIFS client MUST have some means to resolve the name of a CIFS server to a transport address, and that a CIFS server MUST register its name with a name resolution service known its clients.

Some examples of name resolution mechanisms include: using the Domain Name System (DNS) [1,2], and using NETBIOS name resolution (see RFC 1001 and RFC 1002 [3,4]). The server name might also be specified as the string form of an IPv4 address in the usual dotted decimal notation, e.g., "157.33.135.101"; in this case, "resolution" consists of converting to the 32 bit IPv4 address.

Which method is used is configuration dependent; the default SHOULD be DNS to encourage interoperability over the Internet.

Note: The name resolution mechanism used may place constraints on the form of the server name; for example, in the case of NETBIOS, the server name must be 15 characters or less, and MUST be upper case.

2.3. Sample Message Flow

The following illustrates a typical message exchange sequence for a client connecting to a user level server, opening a file, reading its data, closing the file, and disconnecting from the server. Note: using the CIFS request batching mechanism (called the "AndX" mechanism), the second to sixth messages in this sequence can be combined into one, so that there are really only three round trips in the sequence. The last trip can be handled asynchronously by the client.

Client Command	Server Response
SMB_COM_NEGOTIATE	Must be the first message sent by a client to the server. Includes a list of SMB dialects supported by the client. Server response indicates which SMB dialect should be used.
SMB_COM_SESSION_SETUP_ANDX	Transmits the user's name and credentials to the server for verification. Successful server response has Uid field set in SMB header used for subsequent SMBs on behalf of this user.
SMB_COM_TREE_CONNECT_ANDX	Transmits the name of the disk share (exported disk resource) the client wants to access. Printer device and interprocess communication devices are outside the scope of this document. Successful server response has Tid field set in SMB header used for subsequent SMBs referring to this resource.
SMB_COM_OPEN_ANDX	Transmits the name of the file, relative to Tid, the client wants to open. Successful server response includes a file id (Fid) the client should supply for subsequent operations on this file.
SMB_COM_READ	Client supplies Tid, Fid, file offset, and number of bytes to read. Successful server response includes the requested file data.
SMB_COM_CLOSE	Client closes the file represented by Tid and Fid. Server responds with success code.
SMB_COM_TREE_DISCONNECT	Client disconnects from resource represented by Tid.

2.4. CIFS Protocol Dialect Negotiation

The first message sent from a CIFS client to a CIFS server must be one whose Command field is SMB_COM_NEGOTIATE. The format of this client request includes an array of NULL terminated strings indicating the dialects of the CIFS protocol which the client supports. The server compares this list against the list of dialects the server supports and returns the index of the chosen dialect in the response message.

2.5. Message Transport

CIFS is transport independent. The CIFS protocol assumes:

- A reliable connection oriented message-stream transport, and makes no higher level attempts to ensure sequenced delivery of messages between the client and server.
- A well known endpoint for the CIFS service, such as a designated port number.
- Some mechanism to detect failures of either the client or server node, and to deliver such an indication to the client or server software so they can clean up state. When a reliable transport connection from a client terminates, all work in progress by that client is terminated by the server and all resources open by that client on the server are closed.

It can run over any transport that meets these requirements. Some transports do not natively meet all the requirements, and a standard encapsulation of CIFS for that transport may need to

be defined. Appendix A defines how to run CIFS over NETBIOS over TCP; Appendix B defines how to run CIFS over TCP.

2.5.1. Connection Management

Once a connection is established, the rules for reliable transport connection dissolution are:

- If a server receives a transport establishment request from a client with which it is already conversing, the server may terminate all other transport connections to that client. This is to recover from the situation where the client was suddenly rebooted and was unable to cleanly terminate its resource sharing activities with the server.
- A server may drop the transport connection to a client at any time if the client is generating malformed or illogical requests. However, wherever possible the server should first return an error code to the client indicating the cause of the abort.
- If a server gets a unrecoverable error on the transport (such as a send failure) the transport connection to that client may be aborted.
- A server may terminate the transport connection when the client has no open resources on the server, however, we recommend that the termination be performed only after some time has passed or if resources are scarce on the server. This will help performance in that the transport connection will not need to be reestablished if activity soon begins anew. Client software is expected to be able to automatically reconnect to the server if this happens.

2.6. Opportunistic Locks

The CIFS protocol includes a mechanism called “opportunistic locks”, or oplocks, that allows the client to lock a file in such a manner that the server can revoke the lock. The purpose of oplocks is to allow file data caching on the client to occur safely. It does this by defining the conditions under which an oplock is revoked.

When a client opens a file it may request an oplock on the file. If the oplock is given the client may safely perform caching. At some point in the future a second client may open the file. The following steps provide an overview of the actions taken in response to the open from the second client:

- The server holds off responding to the open from the second client.
- The server revokes the oplock of the first client.
- The first client flushes all cached data to the server.
- The first client acknowledges the revoke of the oplock.
- The server responds to the open from the second client.

As can be seen from the above steps, the first client has the opportunity to write back data and acquire record locks before the second client is allowed to examine the file. Because of this a client that holds an oplock can aggressively cache file data and state.

Anecdotal evidence suggests that oplocks provide a performance boost in many real-world applications running on existing CIFS client implementations while preserving data integrity.

2.6.1. Oplock Types

There are three types of oplocks:

- Exclusive
- Batch
- Level II

Versions of the CIFS file sharing protocol including and newer than the "LANMAN1.0" dialect support oplocks. Level II oplocks were introduced in NTLM 0.12.

2.6.1.1. Exclusive and Batch Oplocks

When a client has an exclusive oplock on a file, it is the only client to have the file open. The exclusive oplock allows the client to safely perform file data read and write caching, metadata caching, and record lock caching. All other operations on the file cannot be safely cached.

The server may revoke the exclusive oplock at any time. The client is guaranteed that the server will revoke the exclusive oplock prior to another client successfully opening the file. This gives the client that holds the oplock the opportunity to write back cached information to the file.

The batch oplock was introduced to allow a client to defer closing a file that was opened and re-opened repetitively by an application. It has the same semantics as the exclusive oplock with the following additional guarantee. The client holding a batch oplock has the additional guarantee that the server will revoke the batch oplock prior to another client successfully making any change to the file.

When a client opens a file it can specify that it wants an exclusive oplock, a batch oplock, or no oplock. Exclusive and batch oplocks can only be obtained as a side effect of a file being opened. The protocol does not support other means to obtain exclusive and batch oplocks.

Oplocks can only be obtained on files. Oplocks are not supported on directories and named pipes. However it is not an error to request an oplock on directories and named pipes. In this case the server must return that no oplock was granted.

The server response to a successful open request includes information about what type of oplock was obtained. A server that does not support oplocks should always return that no oplock was granted.

A client that requests an exclusive oplock will get one of the following:

- An exclusive oplock
- A level II oplock
- No oplock

A client that requests a batch oplock will get one of the following:

- A batch oplock
- A level II oplock
- No oplock

A client that requests no oplock will always get no oplock.

The following diagrams the behavior of various clients and the server when an exclusive oplock is obtained on a file and subsequently revoked. The diagram also applies to a batch oplock.

Exclusive/Batch Protocol Oplock Example

Client A	Client B	< -- >	Server
Open file "foo"		- >	
		< -	Open response. Open succeeded. Exclusive oplock granted
Read data		- >	
		< -	Read response with data
Write data (cache)			
Read data (cache)			
	Open file "foo"	- >	
		< -	Oplock break to Client A
Write data		- >	
		< -	Write response
Discard cached data			
Release oplock		- >	
		< -	Open response to B. Open succeeded. No oplock granted.

The revoking of an exclusive or batch oplock involves the server sending an oplock break message to the client, followed by the client flushing file information to the server, followed by the client releasing the oplock. If the client does not respond by releasing the oplock within a period of time acceptable to the server, then the server may consider the oplock released and allow pending operations to proceed. The protocol does not define the duration of the time out period.

When a client opens a file that already has an exclusive oplock, the server first checks the share mode on the file. If the sharing allows the client open to succeed then the exclusive oplock is broken, after which the open is allowed to proceed.

When a client opens a file that already has a batch oplock, the server first revokes the batch oplock. Then the open is allowed to proceed. The reason for this server behavior is that it gives the holder of the oplock the opportunity to close the file. This in turn allows the open to obtain an exclusive or batch oplock.

When a client opens a file that has a security descriptor, the server first checks if the open for the desired access is allowed by the security descriptor. If access is not allowed, the open fails. Any exclusive or batch oplock on the file is not disturbed. Because of this behavior a client holding an exclusive or batch oplock cannot safely cache security descriptor information

2.6.1.2. Level II Oplocks

When a client has a level II oplock on a file, it is an indication to the client that other clients may also have the file open. The level II oplock allows the client to safely perform file data read caching. All other operations on the file cannot be safely cached.

The server may revoke the level II oplock at any time. The client is guaranteed that the server will revoke the level II oplock prior to another client successfully writing the file. This gives the client that holds the level II oplock the opportunity to discard its cached data.

Note however that the level II oplock is revoked differently than an exclusive or batch oplock. A level II oplock break is sent to the client, but a response from the client is not expected. The server allows the write to proceed immediately after the level II oplock break is sent to the client.

A client cannot explicitly request that a level II oplock be granted. A level II oplock is granted either when a file is opened or when a server revokes an exclusive or batch oplock.

When a file is opened the client may request an exclusive or batch oplock. The server has the option of granting a level II oplock instead of the requested type of oplock. This is the only way to obtain a level II oplock when a file is opened.

When a server revokes an exclusive or batch oplock, it may indicate to the client that in conjunction with the revocation that the client is being granted a level II oplock.

The following diagrams the behavior of various clients and the server when a level II oplock is obtained on a file and subsequently revoked.

Level II Oplock Protocol Example

Client A	Client B	< -- >	Server
Open file "foo"		- >	
		< -	Open response. Open succeeded. Exclusive oplock granted
Read data		- >	
		< -	Read response with data
	Open file "foo"	- >	
		< -	Oplock break to Client A. Oplock downgraded to level II.
Release oplock to level II		- >	
		< -	Open response to B. Open succeeded. Oplock level II granted.

2.6.2. Comparison with Other File Locking Methods

The CIFS protocol has three mechanisms to enable a client to control how other clients access a file.

- Opportunistic locks
- Byte range locks
- Sharing locks

Of the three, the server may revoke only opportunistic locks. Byte range and sharing locks are held for as long as the client desires.

Historically on client systems, byte range and sharing locks are exposed to the application. This allows the application to have explicit control over the obtaining and releasing of these types of locks.

Typically however oplocks are not exposed to the application. They are implemented inside the client operating system. The client operating system decides when it is appropriate to obtain and release oplocks. It also handles all of the issues related to revoking of oplocks by the server.

2.6.3. Oplock SMBs

This section summarizes the SMB commands that affect oplocks.

2.6.3.1. Obtaining an Oplock

The following SMB commands may be used to obtain an oplock:

- SMB_COM_OPEN
- SMB_COM_CREATE
- SMB_COM_CREATE_NEW
- SMB_COM_OPEN_ANDX
- SMB_COM_TRANSACTION2 (OPEN2)
- SMB_COM_NT_CREATE_ANDX
- SMB_COM_NT_TRANSACT (NT_CREATE)

The server may only grant a level II oplock to a client for a file when that file is opened using one of “SMB_COM_NT_CREATE_ANDX” or “SMB_COM_NT_TRANSACT (NT_CREATE)”.

2.6.3.2. Releasing an Oplock

A client releases an oplock with the SMB_COM_LOCKING_ANDX command. Alternatively the client may release the oplock by closing the file with the SMB_COM_CLOSE command. Any operation that would invalidate the file handle results in the oplock being released. This includes disconnecting the tree, logging off the user that opened the file, and any action that would disconnect the session.

A client should release its exclusive or batch oplock on a file in response to the server revoking the oplock. Failure to do so is a violation of the protocol.

A client does not need to release a level II oplock (i.e. respond to the server) on a file in response to the server revoking the oplock. However doing so is not an error.

2.6.3.3. Revoking an Oplock

The server revokes a client’s oplock by sending a SMB_COM_LOCKING_ANDX command to the client. The command is sent asynchronously sent from the server to the client. This message has the LOCKING_ANDX_OPLOCK_RELEASE flag set indicating to the client that the oplock is being broken. *OplockLevel* indicates the type of oplock the client now owns. If *OplockLevel* is 0, the client possesses no oplocks on the file at all. If *OplockLevel* is 1, the client possesses a Level II oplock. The client is expected to flush any dirty buffers to the server, submit any file locks, and respond to the server with either an SMB_LOCKING_ANDX SMB having the LOCKING_ANDX_OPLOCK_RELEASE flag set, or with a file close if the file is no longer in use by the client.

2.6.4. Other Issues

Since a close being sent to the server and break oplock notification from the server could cross on the wire, if the client gets an oplock notification on a file that it does not have open, that notification should be ignored. The client is guaranteed that an oplock break notification will not be issued before the server has sent the response to the file open.

Due to timing, the client could get an "oplock broken" notification in a user's data buffer as a result of this notification crossing on the wire with an SMB_COM_READ_RAW request. The client must detect this (use length of message, "FFSMB," MID of -1 and *Command* of SMB_COM_LOCKING_ANDX) and honor the "oplock broken" notification as usual. The server must also note on receipt of an SMB_COM_READ_RAW request that there is an outstanding (unanswered) "oplock broken" notification to the client; it must then return a zero length response denoting failure of the read raw request. The client should (after responding to the "oplock broken" notification) use a non-raw read request to redo the read. This allows a file to actually contain data matching an "oplock broken" notification and still be read correctly.

When an exclusive or batch oplock is being revoked, more than one client open request may be paused until the oplock is released. Once the oplock is released, the order that the paused open requests are processed is not defined.

The protocol allows a client to obtain an oplock and then issue an operation that causes the oplock to be revoked. An example of this is a client obtaining an exclusive oplock on a file and then opening the file a second time.

The protocol allows a client to have a file open multiple times, and each open could have a level II oplock associated with it. A server may choose not to support this situation by simply not handing out more than one level II oplock for a particular file to a particular client.

The protocol allows a server to grant on a single file a level II oplock for some opens and no oplock for other opens. A server may have heuristics that indicate some file opens would not benefit from a level II oplock.

A server that supports access to files via mechanisms other than this protocol must revoke oplocks as necessary to preserve the semantics expected by the clients owning the oplocks.

A client that has an exclusive or batch oplock on a file may cache file metadata. This includes the following information: create time, modify time, access time, change time, file size, file attributes, and extended attributes size. However a server is not required to break an oplock when a second client examines file metadata. Clients should be aware of this behavior when examining file metadata without having the file open.

When a server revokes an exclusive or batch oplock it may grant a level II oplock in its place. The client should consider the level II oplock in effect after the client has released the exclusive or batch oplock. The server may decide to revoke the level II oplock before the client has released the exclusive or batch oplock. In this situation the client should behave as if the revoke of the level II oplock arrived just after the exclusive or batch oplock was released.

2.7. Security Model

Each server makes a set of resources available to clients on the network. A resource being shared may be a directory tree, printer, etc. So far as clients are concerned, the server has no storage or service dependencies on any other servers; a client considers the server to be the sole provider of the file (or other resource) being accessed.

The CIFS protocol requires server authentication of users before file accesses are allowed, and each server authenticates its own users. A client system must send authentication information to the server before the server will allow access to its resources.

A server requires the client to provide a user name and some proof of identity (often something cryptographically derived from a password) to gain access. The granularity of authorization is up to the server. For example, it may use the account name to check access control lists on individual files, or may have one access control list that applies to all files in the directory tree.

When a server validates the account name and password presented by the client, an identifier representing that authenticated instance of the user is returned to the client in the Uid field of the response SMB. This Uid must be included in all further requests made on behalf of the user from that client.

2.8. Authentication

This section defines the CIFS user and message authentication protocols. User authentication allows the server to verify that the client knows a password for a user. Message authentication allows messages in a session to be verified by both the server and the client.

2.8.1. Overview

User authentication is based on the shared knowledge of the user's password. There are two styles of user authentication. The first involves the client sending passwords in plain text to the server. The second involves a challenge/response protocol.

Plain text password authentication exposes the user's password to programs that have access to the CIFS protocol data on the network. For this reason plain text password authentication is discouraged and by default should be disabled in CIFS protocol implementations.

With the challenge/response protocol the server sends a "challenge" to the client, which the client responds to in a way that proves it knows the user's password. A "response" is created from the challenge by encrypting it with a 168 bit "session key" computed from the user's password. The response is then returned to the server, which can validate the response by performing the same computation.

The user authentication protocol is described as if the CIFS server keeps a client's password. However an implementation might actually store the passwords on a key distribution server and have servers use a protocol outside the scope of this document to enable them to perform the steps required by this protocol.

Messages may be authenticated by computing a message authentication code (MAC) for each message and attaching it to the message. The MAC used is a keyed MD5 construction similar to that used in IPSec [RFC 1828], using a "MAC key" computed from the session key, and the response to the server's challenge. The MAC is over both the message text and an implicit sequence number, to prevent replay attacks.

2.8.2. Base Algorithms

Following are definitions of algorithms used by the authentication algorithms.

E(K, D)
: denote the DES block mode encryption function [FIPS 81] , which accepts a seven byte key (K) and an eight byte data block (D) and produces an eight byte encrypted data block as its value.

Ex(K,D)
denote the extension of DES to longer keys and data blocks. If the data to be encrypted is longer than eight bytes, the encryption function is applied to each block of eight bytes in sequence and the results are concatenated together. If the key is longer than seven bytes, each 8 byte block of data is first completely encrypted using the first seven bytes of the key, then the second seven bytes, etc., appending the results each time. For example, to encrypt the 16 byte quantity D0D1 with the 14 byte key K0K1,

Ex(K0K1,D0D1) = concat(E(K0,D0),E(K0,D1),E(K1,D0),E(K1,D1))

concat(A, B, …, Z)
is the result of concatenating the byte strings A, B, … Z

head(S, N)
denote the first N bytes of the byte string S.

swab(S)
denote the byte string obtained by reversing the order of the bits in each byte of S, i.e., if S is byte string of length one, with the value 0x37 then swab(S) is 0xEC.

zeros(N)
denote a byte string of length N whose bytes all have value 0 (zero).

ones(N)
denote a byte string of length N whose bytes all have value 255.

xor(A, B)
denote a byte string formed by the bytewise logical "xor" of each of the bytes in A and B.

and(A, B)
denote a byte string formed by the bytewise logical "and" of each of the bytes in A and B.

substr(S, A, B)
denote a byte string of length N obtained by taking N bytes of S starting at byte A. The first byte is numbered zero. I.e., if S is the string "NONCE" then substr(S, 0, 2) is "NO".

2.8.3. Authentication Algorithms

Following are definitions of the authentication algorithms.

2.8.3.1. NT Session Key

The session key S21 and partial MAC key S16 are computed as

S16 = MD4(PN)

S21 = concat(S16, zeros(5))

where

- PN is a Unicode string containing the user's password in clear text, case sensitive, no maximum length
- MD4(x) of an byte string "x" is the 16 byte MD4 message digest [RFC 1320] of that string

2.8.3.2. LM Session Key

The session key S21 and partial MAC key S16 are computed as

S16X = Ex(swab(P14),N8)

S21 = concat(S16X, zeros(5))

S16 = concat(head(S16X, 8), zeros(8))

Where

- P14 is a 14 byte ASCII string containing the user's password in clear text, upper cased, padded with nulls
- N8 is an 8 byte string whose value is {0x4b, 0x47, 0x53, 0x21, 0x40, 0x23, 0x24, 0x25}

2.8.3.3. Response

The response to the challenge RN is computed as

RN = EX(S21, C8)

Where

- C8 is a 8 byte challenge selected by the server
- S21 is the LM session key or NT session key as determined above

2.8.3.4. MAC key

The MAC key is computed as follows:

K = concat(S16, RN)

Where

- S16 is the partial MAC key computed with the LM session key or NT session key as determined above
- RN is the response to the challenge as determined above
- The result K is either 40 or 44 bytes long, depending on the length of RN. [ed: what determines length of RN?]

2.8.3.5. Message Authentication Code

The MAC is the keyed MD5 construction:

MAC(K, text) = head(MD5(concat(K, text)), 8)

Where

- MD5 is the MD5 hash function; see RFC 1321
- K is the MAC key determined above
- text is the message whose MAC is being computed.

2.8.4. Session Authentication Protocol

2.8.4.1. Plain Text Password

If plaintext password authentication was negotiated, clients send the plaintext password in `SMB_COM_TREE_CONNECT`, `SMB_COM_TREE_CONNECT_ANDX`, and/or `SMB_COM_SESSION_SETUP_ANDX`. The SMB field used to contain the response depends upon the request:

- *Password* in SMB_COM_TREE_CONNECT
- *Password* in SMB_COM_TREE_CONNECT_ANDX
- *AccountPassword* in SMB_COM_SESSION_SETUP_ANDX in dialects prior to "NTLM 0.12"
- *CaseInsensitivePassword* in SMB_COM_SESSION_SETUP_ANDX in the "NTLM 0.12" dialect
- *CaseSensitivePassword* in SMB_COM_SESSION_SETUP_ANDX in the "NTLM 0.12" dialect

2.8.4.2. Challenge/Response

The challenge C8 from the server to the client is contained in the *EncryptionKey* field in the `SMB_COM_NEGPROT` response. Clients send the response to the challenge in `SMB_COM_TREE_CONNECT`, `SMB_COM_TREE_CONNECT_ANDX`, and/or `SMB_COM_SESSION_SETUP_ANDX`. The SMB field used to contain the response depends upon the request:

- *Password* in SMB_COM_TREE_CONNECT
- *Password* in SMB_COM_TREE_CONNECT_ANDX
- *AccountPassword* in SMB_COM_SESSION_SETUP_ANDX in dialects prior to "NTLM 0.12"
- *CaseInsensitivePassword* in SMB_COM_SESSION_SETUP_ANDX for a response computed using the "LM session key" in the "NTLM 0.12" dialect
- *CaseSensitivePassword* in SMB_COM_SESSION_SETUP_ANDX for a response computed using the "NT session key" in the "NTLM 0.12" dialect

The challenge/response authentication protocol has the following steps:

- The server chooses an 8 byte challenge C8 and sends it to the client.
- The client computes RN as described above
- The client sends the 24 byte response RN to the server
- The server computes RN as described above and compares the received response with its computed value for RN; if equal, the client has authenticated.

2.8.5. Message authentication code

Once a user logon has been authenticated, each message can be authenticated as well. This will prevent man in the middle attacks, replay attacks, and active message modification attacks.

To use message authentication, the client sets `SMB_FLAGS2_SMB_SECURITY_SIGNATURE` in SMB_COM_SESSION_SETUP_ANDX request to the server, and includes a MAC. If the resulting

logon is non-null and non-guest, then the SMB_COM_SESSION_SETUP_ANDX response and all subsequent SMB requests and responses must include a MAC. The first non-null, non-guest logon determines the key to be used for the MAC for all subsequent sessions.

Message authentication may only be requested when the "NTLM 0.12" dialect has been negotiated. If message authentication is used, raw mode MUST not be used (because some raw mode messages have no headers in which to carry the MAC).

Let

- SN be a request sequence number, initially set to 0. Both client and server have one SN for each connection between them.
- RSN be the sequence number expected on the response to a request.
- req_msg be a request message
- rsp_msg be a response message

The SN is logically contained in each message and participates in the computation of the MAC.

For each message sent in the session, the following procedure is followed:

- Client computes MAC(req_msg) using SN, and sends it to the server in the request message. If there are multiple requests in the message (using the "AndX" facility), then the MAC is calculated as if it were a single large request.
- Client increments its SN and saves it as RSN
- Client increments its SN – this is the SN it will use in its next request
- Server receives each req_msg, validates MAC(req_msg) using SN, and responds ACCESS_DENIED if invalid
- Server increments its SN and saves it as RSN
- Server increments its SN – this is the SN it will expect in the next request
- Server computes MAC(rsp_msg) using RSN, and sends it to client in the response message. If there are multiple responses in the message (using the "AndX" facility) , then the MAC is calculated as if it were a single large response.
- Client receives each rsp_msg, validates MAC(rsp_msg) using RSN, and discards the response message if invalid

In each message that contains a MAC, the following bit is set in the flags2 field:

```
#define SMB_FLAGS2_SMB_SECURITY_SIGNATURES 0x0004
```

The sender of a message inserts the sequence number SSN into the message by putting it into the first 4 bytes of the SecuritySignature field and zeroing the last 4 bytes, computes the MAC over the entire message, then puts the MAC in the field. The receiver of a message validates the MAC by extracting the value of the SecuritySignature field, putting its ESN into the first 4 bytes of the SecuritySignature field and zeroing the last 4 bytes, computing the MAC, and comparing it to the extracted value.

Oplock break messages from the server to the client may not use message authentication, even if it has been negotiated.

2.8.6. Security Level

The SMB_COM_NEGPROT response from a server has the following bits in its *SecurityMode* field:

```
#define NEGOTIATE_SECURITY_USER_LEVEL                0x01
#define NEGOTIATE_SECURITY_CHALLENGE_RESPONSE        0x02
#define NEGOTIATE_SECURITY_SIGNATURES_ENABLED        0x04
#define NEGOTIATE_SECURITY_SIGNATURES_REQUIRED       0x08
```

If NEGOTIATE_SECURITY_USER_LEVEL is set, then "user level" security is in effect for all the shares on the server. This means that the client must establish a logon (with SMB_COM_SESSION_SETUP_ANDX) to authenticate the user before connecting to a share, and the password to use in the authentication protocol described above is the user's password. If NEGOTIATE_SECURITY_USER_LEVEL is clear, then "share level" security is in effect for all the shares in the server. In this case the authentication protocol is a password for the share.

If NEGOTIATE_SECURITY_CHALLENGE_RESPONSE is clear, then the server is requesting plaintext passwords.

If NEGOTIATE_SECURITY_CHALLENGE_RESPONSE is set, then the server supports the challenge/response session authentication protocol described above, and clients should use it. Servers may refuse connections that do not use it.

If the dialect is earlier than "NTLM 0.12" then the client computes the response using the "LM session key". If the dialect is "NTLM 0.12" then the client may compute the response either using the "LM session key", or the "NT session key", or both. The server may choose to refuse responses computed using the "LM session key".

If NEGOTIATE_SECURITY_SIGNATURES_ENABLED is set, then the server supports the message authentication protocol described above, and the client may use it. This bit may only be set if NEGOTIATE_SECURITY_CHALLENGE_RESPONSE is set.

If NEGOTIATE_SECURITY_SIGNATURES_REQUIRED is set, then the server requires the use of the message authentication protocol described above, and the client must use it. This bit may only be set if NEGOTIATE_SECURITY_SIGNATURES_ENABLED is set. This bit must not be set if NEGOTIATE_SECURITY_USER_LEVEL is clear (i.e., for servers using "share level" security).

2.9. Distributed File System (DFS) Support

Protocol dialects of NT LM 0.12 and later support distributed filesystem operations. The distributed filesystem gives a way for this protocol to use a single consistent file naming scheme which may span a collection of different servers and shares. The distributed filesystem model employed is a referral - based model. This protocol specifies the manner in which clients receive referrals.

The client can set a flag in the request SMB header indicating that the client wants the server to resolve this SMB's paths within the DFS known to the server. The server attempts to resolve the requested name to a file contained within the local directory tree indicated by the TID of the request and proceeds normally. If the request pathname resolves to a file on a different system, the server returns the following error:

> STATUS_DFS_PATH_NOT_COVERED - the server does not support the part of the DFS namespace needed to resolve the pathname in the request. The client should request a referral from this server for further information.

A client asks for a referral with the TRANS2_DFS_GET_REFERRAL request containing the DFS pathname of interest. The response from the server indicates how the client should proceed.

The method by which the topological knowledge of the DFS is stored and maintained by the servers is not specified by this protocol.

3. SMB Message Formats and Data Types

Clients exchange messages with a server to access resources on that server. These messages are called Server Message Blocks (SMBs), and every SMB message has a common format.

This section describes the entire set of SMB commands and responses exchanged between CIFS clients and servers. It also details which SMBs are introduced into the protocol as higher dialect levels are negotiated.

3.1. Notation

This document makes use of "C"-like notation to describe the formats of messages. Unlike the "C" language, which allows for implementation flexibility in laying out structures, this document adopts the following rules. Multi-byte values are always transmitted least significant byte first. All fields, except "bit-fields", are aligned on the nearest byte boundary (even if longer than a byte), and there is no implicit padding. Fields using the "bit field" notation are defined to be laid out within the structure with the first-named field occupying the lowest order bits, the next named field the next lowest order bits, and so on. BOOLEAN is defined to be a single byte. The SHORT and LONG types are little endian.

3.2. SMB header

While each SMB command has specific encodings, there are some fields in the SMB header, which have meaning to all SMBs. These fields and considerations are described in the following sections.

```
typedef unsigned char UCHAR;           // 8 unsigned bits
typedef unsigned short USHORT;         // 16 unsigned bits
typedef unsigned long ULONG;           // 32 unsigned bits

typedef struct {
    ULONG LowPart;
    LONG HighPart;
} LARGE_INTEGER;                       // 64 bits of data

typedef struct  {
    UCHAR Protocol[4];                 // Contains 0xFF,'SMB'
    UCHAR Command;                     // Command code
    union {
        struct {
            UCHAR ErrorClass;          // Error class
            UCHAR Reserved;            // Reserved for future use
            USHORT Error;              // Error code
        } DosError;
        ULONG Status;                  // 32-bit error code
    } Status;
    UCHAR Flags;                       // Flags
    USHORT Flags2;                     // More flags
    union {
        USHORT Pad[6];                 // Ensure section is 12 bytes long
        struct {
            USHORT PidHigh;           // High Part of PID
            UCHAR SecuritySignature[8];   // reserved for MAC
       } Extra;
    };
```

```
    USHORT Tid;                         // Tree identifier
    USHORT Pid;                         // Caller's process ID, opaque for
client use
    USHORT Uid;                         // User id
    USHORT Mid;                         // multiplex id
    UCHAR  WordCount;                   // Count of parameter words
    USHORT ParameterWords[WordCount];     // The parameter words
    USHORT ByteCount;                   // Count of bytes
    UCHAR  Buffer[ByteCount];           // The bytes
} SMB_HEADER;
```

All SMBs in this document have an identical format up to the ParameterWords field. (However, this is not true for some obsolescent SMBs.) For the last fields in the header, different SMBs have a different number and interpretation of the ParameterWords and Buffer fields. All reserved fields in the SMB header must be zero.

3.2.1. Command field

The Command is the operation code that this SMB is requesting or responding to. See section 5.1 below for number values, and section 4 for a description of each operation.

3.2.2. Flags field

This field contains 8 individual flags, numbered from least significant bit to most significant bit, which are defined below. Flags that are not defined MUST be set to zero by clients and MUST be ignored by servers.

Bit	Meaning	Earliest Dialect
0	Reserved for obsolescent requests LOCK_AND_READ, WRITE_AND_CLOSE	LANMAN1.0
1	Reserved (must be zero).	
2	Reserved (must be zero).	
3	When on, all pathnames in this SMB must be treated as case-less. When off, the pathnames are case sensitive.	LANMAN1.0
4	Obsolescent – client case maps (canonicalizes) file and directory names; servers must ignore this flag.	
5	Reserved for obsolescent requests – oplocks supported for SMB_COM_OPEN, SMB_COM_CREATE and SMB_COM_CREATE_NEW. Servers must ignore when processing all other SMB commands.	LANMAN1.0
6	Reserved for obsolescent requests – notifications supported for SMB_COM_OPEN, SMB_COM_CREATE and SMB_COM_CREATE_NEW. Servers must ignore when processing all other SMB commands.	LANMAN1.0
7	SMB_FLAGS_SERVER_TO_REDIR - When on, this SMB is being sent from the server in response to a client request. The Command field usually contains the same value in a protocol request from the client to the server as in the matching response from the server to the client. This bit unambiguously distinguishes the command request from the command response.	PC NETWORK PROGRAM 1.0

3.2.3. Flags2 Field

This field contains nine individual flags, numbered from least significant bit to most significant bit, which are defined below. Flags that are not defined MUST be set to zero by clients and MUST be ignored by servers.

Bit	Name: SMB_FLAGS2_	Meaning	Earliest Dialect
0	KNOWS_LONG_NAMES	If set in a request, the server may return long components in path names in the response.	LM1.2X002
1	KNOWS_EAS	If set, the client is aware of extended attributes (EAs).	
2	SECURITY_SIGNATURE	If set, the SMB is integrity checked.	
3	RESERVED1	Reserved for future use	
6	IS_LONG_NAME	If set, any path name in the request is a long name.	
11	EXT_SEC	If set, the client is aware of Extended Security negotiation.	NT LM 0.12
12	DFS	If set, any request pathnames in this SMB should be resolved in the Distributed File System.	NT LM 0.12
13	PAGING_IO	If set, indicates that a read will be permitted if the client does not have read permission but does have execute permission. This flag is only useful on a read request.	
14	ERR_STATUS	If set, specifies that the returned error code is a 32 bit error code in Status.Status. Otherwise the Status.DosError.ErrorClass and Status.DosError.Error fields contain the DOS-style error information. When passing NT status codes is negotiated, this flag should be set for every SMB.	NT LM 0.12
15	UNICODE	If set, any fields of datatype STRING in this SMB message are encoded as UNICODE. Otherwise, they are in ASCII. The character encoding for Unicode fields SHOULD be UTF-16 (little endian).	NT LM 0.12

3.2.4. Tid Field

Tid represents an instance of an authenticated connection to a server resource. The server returns Tid to the client when the client successfully connects to a resource, and the client uses Tid in subsequent requests referring to the resource.

In most SMB requests, Tid must contain a valid value. Exceptions are those used prior to getting a Tid established, including SMB_COM_NEGOTIATE, SMB_COM_TREE_CONNECT_ANDX, SMB_COM_ECHO, and SMB_COM_SESSION_SETUP_ANDX. 0xFFFF should be used for Tid for these situations. The server is always responsible for enforcing use of a valid Tid where appropriate.

On SMB_COM_TREE_DISCONNECT over a given transport connection, with a given Tid, the server will close any files opened with that Tid over that connection.

3.2.5. Pid Field

Pid is the caller's process id, and is generated by the client to uniquely identify a process within the client computer. Concurrency control is associated with Pid (and PidHigh)—sharing modes, and locks are arbitrated using the Pid. For example, if a file is successfully opened for exclusive access, subsequent opens from other clients or from the same client with a different Pid will be refused.

Clients inform servers of the creation of a new process by simply introducing a new Pid value into the dialogue for new processes. The client operating system must ensure that the appropriate close and cleanup SMBs will be sent when the last process referencing a file closes it. From the server's point of view, there is no concept of Fids "belonging to" processes. A Fid returned by the server to one process may be used by any other process using the same transport connection and Tid.

It is up to the client operating system to ensure that only authorized client processes gain access to Fids (and Tids). On SMB_COM_TREE_DISCONNECT (or when the client and server session is terminated) with a given Tid, the server will invalidate any files opened by any process on that client.tid Field

3.2.6. Uid Field

Uid is a reference number assigned by the server after a user authenticates to it, and that it will associate with that user until the client requests the association be broken. After authentication to the server, the client SHOULD make sure that the Uid is not used for a different user that the one that authenticated. (It is permitted for a single user to have more than one Uid.) Requests that do authorization, such as open requests, will perform access checks using the identity associated with the Uid.

3.2.7. Mid Field

The multiplex ID (Mid) is used along with the Pid to allow multiplexing the single client and server connection among the client's multiple processes, threads, and requests per thread. Clients may have many outstanding requests (up to the negotiated number, MaxMpxCount) at one time. Servers MAY respond to requests in any order, but a response message MUST always contain the same Mid and Pid values as the corresponding request message. The client MUST NOT have multiple outstanding requests to a server with the same Mid and Pid.

3.2.8. Status Field

An SMB returns error information to the client in the Status field. Protocol dialects prior to NT LM 0.12 return status to the client using the combination of Status.DosError.ErrorClass and Status.DosError.Error. Beginning with NT LM 0.12 CIFS servers can return 32 bit error information to clients using Status.Status if the incoming client SMB has bit 14 set in the Flags2 field of the SMB header. The contents of response parameters are not guaranteed in the case of an error return, and must be ignored. For write-behind activity, a subsequent write or close of the file may return the fact that a previous write failed. Normally write-behind failures are limited to hard disk errors and device out of space.

3.2.9. Timeouts

In general, SMBs are not expected to block at the server; they should return "immediately". There are however a series of operations which may block for a significant time. The most obvious of these is named-pipe operations, which may be dependent on another application completing a

write before they can fully complete their read. (Most named-pipe operations are never expired unless cancelled). Similarly, with byte-range locking, the Timeout period is specified by the client, so the server is not responsible for blocking on this operation as long as the client has specified it may. A SMB server should put forth its best effort to handle operations as they arrive in an efficient manner, such that clients do not timeout operations believing the server to be unresponsive falsely. A client may timeout a pending operation by terminating the session. If a server implementation can not support timeouts, then an error can be returned just as if a timeout had occurred if the resource is not available immediately upon request.

3.2.10. Data Buffer (BUFFER) and String Formats

The data portion of SMBs typically contains the data to be read or written, file paths, or directory paths. The format of the data portion depends on the message. All fields in the data portion have the same format. In every case it consists of an identifier byte followed by the data.

Identifier	Description	Value
Data Block	See below	1
Dialect	Null terminated string	2
Pathname	Null terminated string	3
ASCII	Null terminated string	4
Variable Block	See below	5

When the identifier indicates a data block or variable block then the format is a word indicating the length followed by the data.

In all dialects prior to NT LM 0.12, all strings are encoded in ASCII. If the agreed dialect is NT LM 0.12 or later, Unicode strings may be exchanged. Unicode strings include file names, resource names, and user names. This applies to null-terminated strings, length specified strings and the type-prefixed strings. In all cases where a string is passed in Unicode format, the Unicode string must be word-aligned with respect to the beginning of the SMB. Should the string not naturally fall on a two-byte boundary, a null byte of padding will be inserted, and the Unicode string will begin at the next address. In the description of the SMBs, items that may be encoded in Unicode or ASCII are labeled as STRING. If the encoding is ASCII, even if the negotiated string is Unicode, the quantity is labeled as UCHAR.

For type-prefixed Unicode strings, the padding byte is found after the type byte. The type byte is 4 (indicating SMB_FORMAT_ASCII) independent of whether the string is ASCII or Unicode. For strings whose start addresses are found using offsets within the fixed part of the SMB (as opposed to simply being found at the byte following the preceding field,) it is guaranteed that the offset will be properly aligned.

Strings that are never passed in Unicode are:

- The protocol strings in the Negotiate SMB request.
- The service name string in the Tree_Connect_AndX SMB.

When Unicode is negotiated, the SMB_FLAGS2_UNICODE bit should be set in the Flags2 field of every SMB header.

Despite the flexible encoding scheme, no field of a data portion may be omitted or included out of order. In addition, neither a WordCount nor ByteCount of value 0 at the end of a message may be omitted.

3.3. Name Restrictions

The following four reserved characters MUST not be used in share names (network names), user names, group names or domain names.

"\", "/", "?", "*"

The following ten characters SHOULD not be used in share names, user names, group names or domain names as they are considered reserved by multiple existing implementations:

"[", "]", ":", ";", "|", "=", ",", "+", "<", ">"

A share name or server or workstation name SHOULD not begin with a period (".") nor should it include two adjacent periods ("..").

The same naming considerations apply for RFC 1001 names for servers or workstations when using Netbios over TCP/IP name resolution mechanisms.

3.4. File Names

File names in the CIFS protocol consist of components separated by a backslash ('\'). Early clients of the CIFS protocol required that the name components adhere to an 8.3 format name. These names consist of two parts: a basename of no more than 8 characters, and an extension of no more than 3 characters. The basename and extension are separated by a '.'. All characters are legal in the basename and extension except the space character (0x20) and:

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

If the client has indicated long name support by setting bit2 in the Flags2 field of the SMB header, this indicates that the client is not bound by the 8.3 convention. Specifically this indicates that any SMB which returns file names to the client may return names which do not adhere to the 8.3 convention, and have a total length of up to 255 characters. This capability was introduced with the LM1.2X002 protocol dialect.

The two special path components "." and ".." MUST be recognized. They indicate the current directory and the parent directory respectively. Although the use of ".." permits the specification of resources "above" the root of the tree connection, servers SHOULD prevent access to files or directories above the root of the exported share.

3.5. Wildcards

Some SMB requests allow wildcards to be given for the filename. The wildcard allows a number of files to be operated on as a unit without having to separately enumerate the files and individually operate on each one from the client. Two different sets of search semantics are supported. DOS search semantics are used for searching by 8.3 (or short names). Normal search semantics are used for searching by long names (those which support file names different from 8.3).

In the 8.3 naming scheme, each file name can contain up to 8 characters, a dot, and up to 3 trailing characters. Each part of the name (base (8) or extension (3)) is treated separately. The "*", the "?" and the "." can be used as wildcards. The "*" matches 0 or more characters until encountering and matching the "." in the name. The "?" matches any single character, or upon encountering a "." or end of name string, advances the expression to the end of the set of contiguous "?"s. So if the filename part commences with one or more "?"s then exactly that number of characters will be matched by the wildcards, e.g., "??x" equals "abx" but not "abcx" or "ax". When a filename part has trailing "?"s then it matches the specified number of characters or less, e.g., "x??" matches "xab", "xa" and "x", but not "xabc". If only "?"s are present in the filename

part, then it is handled as for trailing “?”s. Finally, the “.” Matches either a “.” or an empty extension string.

In the normal naming scheme, the “.” In the name is significant even though there is no longer a restriction on the size of each of the file name components. A file name may have none, one or more than one “.”s within its name. Spaces “ “ are also allowed within file names and both follow normal wildcard searching rules. For example, if the files “foo bar none” and “foo.bar.none” exist, the pattern “foo*” equals both, “foo.*” equals “foo.bar.none” and “foo *” equals “foo bar none”.

The ? character is a wildcard for a single character. If the match pattern commences with one or more "?"s then exactly that number of characters will be matched by the wildcards, e.g., "??x" equals "abx" but not "abcx" or "ax". When a match pattern has trailing "?"s then it matches the specified number of characters or less, e.g., "x??" matches "xab", "xa" and "x", but not "xabc". If only "?"s are present in the match pattern, then it is handled as for trailing "?"s.

The * character matches an entire name. For example, "*" matches all files in a directory.

If the negotiated dialect is "NT LM 0.12" or later, and the client requires MS-DOS wildcard matching semantics, UNICODE wildcards should be translated according to the following rules:

- Translate the “?” literal to “>”
- Translate the “.” literal to “”” if it is followed by a “?” or a “*”
- Translate the “*” literal to “<” if it is followed by a “.”

The translation can be performed in-place.

3.6. DFS Pathnames

A DFS pathname adheres to the standard described in the FileNames section. A DFS enabled client accessing a DFS share should set the Flags2 bit 12 in all name based SMB requests indicating to the server that the enclosed pathname should be resolved in the Distributed File System namespace. The pathname should always have the full file name, including the server name and share name. If the server can resolve the DFS name to a piece of local storage, the local storage will be accessed. If the server determines that the DFS name actually maps to a different server share, the access to the name will fail with the 32-bit status STATUS_PATH_NOT_COVERED (0xC0000257), or DOS error ERRsrv/ERRbadpath.

On receiving this error, the DFS enabled client should ask the server for a referral (see TRANS2_GET_DFS_REFERRAL). The referral request should contain the full file name.

The response to the request will contain a list of server and share names to try, and the part of the request file name that junctions to the list of server shares. If the ServerType field of the referral is set to 1 (SMB server), then the client should resubmit the request with the original file name to one of the server shares in the list, once again setting the Flags2 bit 12 bit in the SMB. If the ServerType field is not 1, then the client should strip off the part of the file name that junctions to the server share before resubmitting the request to one of servers in the list.

A response to a referral request may elicit a response that does not have the StorageServers bit set. In that case, the client should resubmit the referral request to one of the servers in the list, until it finally obtains a referral response that has the StorageServers bit set, at which point the client can resubmit the request SMB to one of the listed server shares.

If, after getting a referral with the StorageServers bit set and resubmitting the request to one of the server shares in the list, the server fails the request with STATUS_PATH_NOT_COVERED, it must be the case that there is an inconsistency between the view of the DFS namespace held by the server granting the referral and the server listed in that referral. In this case, the client may

inform the server granting the referral of this inconsistency via the TRANS2_REPORT_DFS_INCONSISTENCY SMB.

3.7. Time And Date Encoding

When SMB requests or responses encode time values, the following describes the various encodings used.

```
struct {
        USHORT Day : 5;
        USHORT Month : 4;
        USHORT Year : 7;
} SMB_DATE;
```

The Year field has a range of 0-119, which represents years 1980 - 2099. The Month is encoded as 1-12, and the day ranges from 1-31.

```
struct {
        USHORT TwoSeconds : 5;
        USHORT Minutes : 6;
        USHORT Hours : 5;
} SMB_TIME;
```

Hours ranges from 0-23, Minutes range from 0-59, and TwoSeconds ranges from 0-29 representing two second increments within the minute.

```
typedef struct {
    ULONG LowTime;
    LONG HighTime;
} TIME;
```

TIME indicates a signed 64-bit integer representing either an absolute time or a time interval. Times are specified in units of 100ns. A positive value expresses an absolute time. The time base (the 64-bit integer with value 0) is the beginning of the year 1601 AD in the Gregorian calendar UTC. However, file creation, modification and access times include an additional correction factor as follows:

```
Tf = Tutc + Tdaf - Tdan
```

Where

```
Tf   time reported for file creation/modification/deletion
Tutc UTC time (secs since 1601 AD)
Tdaf Daylight savings adjustment (positive quantity) in effect at Tf
Tdan Current daylight savings adjustment (positive quantity)
```

For example, if a file is created in the summer - when daylight savings time is in effect - the creation time will be reported as

```
Summer:  Tutc + 3600 - 3600 = Tutc
Winter:  Tutc + 3600 - 0    = Tutc + 3600
```

If a file is created during the winter - when daylight savings time not in effect - the creation time will be reported as:

```
Summer:  Tutc + 0 - 3600 = Tutc - 3600
Winter:  Tutc + 0 - 0    = Tutc
```

A negative value expresses a time interval relative to some base time, usually the current time.

```
typedef unsigned long UTIME;
```

UTIME is the number of seconds since Jan 1, 1970, 00:00:00.0.

3.8. Access Mode Encoding

Various client requests and server responses, such as SMB_COM_OPEN, pass file access modes encoded into a USHORT. The encoding of these is as follows:

```
1111 11
5432 1098 7654 3210
rWrC rLLL rSSS rAAA
```

where:

```
W - Write through mode.  No read ahead or write behind allowed on
    this file or device.  When the response is returned, data is
    expected to be on the disk or device.
S - Sharing mode:
    0 - Compatibility mode
    1 - Deny read/write/execute (exclusive)
    2 - Deny write
    3 - Deny read/execute
    4 - Deny none
A - Access mode
    0 - Open for reading
    1 - Open for writing
    2 - Open for reading and writing
    3 - Open for execute

rSSSrAAA = 11111111 (hex FF) indicates FCB open (???)

C - Cache mode
    0 - Normal file
    1 - Do not cache this file
L - Locality of reference
    0 - Locality of reference is unknown
    1 - Mainly sequential access
    2 - Mainly random access
    3 - Random access with some locality
    4 to 7 - Currently undefined
```

3.9. Access Mask Encoding

The ACCESS_MASK structure is one 32-bit value containing standard, specific, and generic rights. These rights are used in access-control entries (ACEs) and are the primary means of specifying the requested or granted access to an object.

The bits in this value are allocated as follows: Bits 0-15 contain the access mask specific to the object type associated with the mask. Bits 16-23 contain the object's standard access rights and can be a combination of the following predefined flags:

Flag	Value	Meaning
DELETE	0x00010000	Delete access
READ_CONTROL	0x00020000	Read access to the owner, group, and discretionary access-control list (ACL) of the security descriptor
WRITE_DAC	0x00040000	Write access to the discretionary access-control list (ACL)

WRITE_OWNER	0x00080000	Write access to owner
SYNCHRONIZE	0x00100000	Windows NT: Synchronize access
STANDARD_RIGHTS_REQUIRED	0x000F0000	
STANDARD_RIGHTS_READ	READ_CONTROL	
STANDARD_RIGHTS_WRITE	READ_CONTROL	
STANDARD_RIGHTS_EXECUTE	READ_CONTROL	
STANDARD_RIGHTS_ALL	0x001F0000	
SPECIFIC_RIGHTS_ALL	0x0000FFFF	
22		
23		
ACCESS_SYSTEM_SECURITY	0x01000000	This flag is not a typical access type. It is used to indicate access to a system ACL. This type of access requires the calling process to have a specific privilege.
MAXIMUM_ALLOWED	0x02000000	
26		Reserved
27		Reserved
GENERIC_ALL	0x10000000	
GENERIC_EXECUTE	0x20000000	
GENERIC_WRITE	0x40000000	
GENERIC_READ	0x80000000	

3.10. Open Function Encoding

OpenFunction specifies the action to be taken depending on whether or not the file exists. This word has the following format:

```
1111 11
5432 1098 7654 3210
rrrr rrrr rrrC rrOO
```

where:

```
C - Create (action to be taken if file does not exist)
    0 -- Fail
    1 -- Create file
r - reserved (must be zero)
O - Open (action to be taken if file exists)
    0 - Fail
    1 - Open file
    2 - Truncate file
```

3.11. Open Action Encoding

Action in the response to an open or create request describes the action taken as a result of the request. It has the following format:

```
1111 11
```

```
5432 1098 7654 3210
Lrrr rrrr rrrr rrOO
```

where:

```
L - Lock (single user total file lock status)
    0 -- file opened by another user (or mode not supported by server)
    1 -- file is opened only by this user at the present time
r - reserved (must be zero)
O - Open (action taken on Open)
    1 - The file existed and was opened
    2 - The file did not exist but was created
    3 - The file existed and was truncated
```

3.12. File Attribute Encoding

When SMB messages exchange file attribute information, it is encoded in 16 bits as:

Value	Description
0x01	Read only file
0x02	Hidden file
0x04	System file
0x08	Volume
0x10	Directory
0x20	Archive file
Others	Reserved – Must be 0

3.13. Extended File Attribute Encoding

The extended file attributes is a 32 bit value composed of attributes and flags.

Any combination of the following attributes is acceptable, except all other file attributes override FILE_ATTR_NORMAL:

Name	Value	Meaning
ATTR_ARCHIVE	0x020	The file has not been archived since it was last modified. Applications use this attribute to mark files for backup or removal.
ATTR_COMPRESSED	0x800	The file or directory is compressed. For a file, this means that all of the data in the file is compressed. For a directory, this means that compression is the default for newly created files and subdirectories. The state of the attribute ATTR_COMPRESSED does not affect how data is read or written to the file or directory using the SMB operations. The attribute only indicates how the server internally stores the data.
ATTR_NORMAL	0x080	The file has no other attributes set. This attribute is valid only if used alone.
ATTR_HIDDEN	0x002	The file is hidden. It is not to be included in an ordinary directory listing.
ATTR_READONLY	0x001	The file is read only. Applications can read the file but cannot write to it or delete it.
ATTR_TEMPORARY	0x100	The file is temporary.
ATTR_DIRECTORY	0x010	The file is a directory.
ATTR_SYSTEM	0x004	The file is part of or is used exclusively by the operating system.

Any combination of the following flags is acceptable:

Name	Value	Meaning
WRITE_THROUGH	0x80000000	Instructs the operating system to write through any intermediate cache and go directly to the file. The operating system can still cache write operations, but cannot lazily flush them.
NO_BUFFERING	0x20000000	Requests the server to open the file with no intermediate buffering or caching; the server is not obliged to honor the request. An application must meet certain requirements when working with files opened with FILE_FLAG_NO_BUFFERING. File access must begin at offsets within the file that are integer multiples of the volume's sector size; and must be for numbers of bytes that are integer multiples of the volume's sector size. For example, if the sector size is 512 bytes, an application can request reads and writes of 512, 1024, or 2048 bytes, but not of 335, 981, or 7171 bytes.
RANDOM_ACCESS	0x10000000	Indicates that the application intends to access the file randomly. The server MAY use this flag to optimize file caching.

Name	Value	Meaning
SEQUENTIAL_SCAN	0x08000000	Indicates that the file is to be accessed sequentially from beginning to end. Windows uses this flag to optimize file caching. If an application moves the file pointer for random access, optimum caching may not occur; however, correct operation is still guaranteed. Specifying this flag can increase performance for applications that read large files using sequential access. Performance gains can be even more noticeable for applications that read large files mostly sequentially, but occasionally skip over small ranges of bytes.
DELETE_ON_CLOSE	0x04000000	Requests that the server is delete the file immediately after all of its handles have been closed.
BACKUP_SEMANTICS	0x02000000	Indicates that the file is being opened or created for a backup or restore operation. The server SHOULD allow the client to override normal file security checks, provided it has the necessary permission to do so.
POSIX_SEMANTICS	0x01000000	Indicates that the file is to be accessed according to POSIX rules. This includes allowing multiple files with names differing only in case, for file systems that support such naming. (Use care when using this option because files created with this flag may not be accessible by applications written for MS-DOS, Windows 3.x, or Windows NT.)

3.14. Batching Requests ("AndX" Messages)

LANMAN1.0 and later dialects of the CIFS protocol allow multiple SMB requests to be sent in one message to the server. Messages of this type are called AndX SMBs, and they obey the following rules:

- The embedded command does not repeat the SMB header information. Rather the next SMB starts at the WordCount field.
- All multiple (chained) requests must fit within the negotiated transmit size. For example, if SMB_COM_TREE_CONNECT_ANDX included SMB_COM_OPEN_ANDX and SMB_COM_WRITE, they would all have to fit within the negotiated buffer size. This would limit the size of the write.
- There is one message sent containing the chained requests and there is one response message to the chained requests. The server may NOT elect to send separate responses to each of the chained requests.
- All chained responses must fit within the negotiated transmit size. This limits the maximum value on an embedded SMB_COM_READ for example. It is the client's responsibility to not request more bytes than will fit within the multiple response.
- The server will implicitly use the result of the first command in the "X" command. For example the Tid obtained via SMB_COM_TREE_CONNECT_ANDX would be used in the embedded SMB_COM_OPEN_ANDX, and the Fid obtained in the SMB_COM_OPEN_ANDX would be used in the embedded SMB_COM_READ.
- Each chained request can only reference the same Fid and Tid as the other commands in the combined request. The chained requests can be thought of as performing a single (multi-part) operation on the same resource.
- The first Command to encounter an error will stop all further processing of embedded commands. The server will not back out commands that succeeded. Thus if a chained request contained SMB_COM_OPEN_ANDX and SMB_COM_READ and the server was

able to open the file successfully but the read encountered an error, the file would remain open. This is exactly the same as if the requests had been sent separately.

- If an error occurs while processing chained requests, the last response (of the chained responses in the buffer) will be the one which encountered the error. Other unprocessed chained requests will have been ignored when the server encountered the error and will not be represented in the chained response. Actually the last valid AndXCommand (if any) will represent the SMB on which the error occurred. If no valid AndXCommand is present, then the error occurred on the first request/response and Command contains the command which failed. In all cases the error information are returned in the SMB header at the start of the response buffer.
- Each chained request and response contains the offset (from the start of the SMB header) to the next chained request/response (in the AndXOffset field in the various "and X" protocols defined later e.g. SMB_COM_OPEN_ANDX). This allows building the requests unpacked. There may be space between the end of the previous request (as defined by WordCount and ByteCount) and the start of the next chained request. This simplifies the building of chained protocol requests. Note that because the client must know the size of the data being returned in order to post the correct number of receives (e.g. SMB_COM_TRANSACTION, SMB_COM_READ_MPX), the data in each response SMB is expected to be truncated to the maximum number of 512 byte blocks (sectors) which will fit (starting at a 32 bit boundary) in the negotiated buffer size with the odd bytes remaining (if any) in the final buffer.

3.15. "Transaction" Style Subprotocols

The "transaction" style subprotocols are used for commands that potentially need to transfer a large amount of data (greater than 64K bytes).

3.15.1. SMB_COM_TRANSACTION2 Format

The following list describes the format of the TRANSACTION2 client request:

```
Primary Client Request          Description
======================          ============
 Command                         SMB_COM_TRANSACTION2
 UCHAR WordCount;                Count of parameter words; value =
                                           (14 + SetupCount)
 USHORT TotalParameterCount;     Total parameter bytes being sent
 USHORT TotalDataCount;          Total data bytes being sent
 USHORT MaxParameterCount;       Max parameter bytes to return
 USHORT MaxDataCount;            Max data bytes to return
 UCHAR MaxSetupCount;            Max setup words to return
 UCHAR Reserved;
 USHORT Flags;                   Additional information:
                                    bit 0 - Disconnect TID
 ULONG Timeout;
 USHORT Reserved2;
 USHORT ParameterCount;          Parameter bytes sent this buffer
 USHORT ParameterOffset;         Offset (from header start) to
                                           Parameters
 USHORT DataCount;               Data bytes sent this buffer
 USHORT DataOffset;              Offset (from header start) to data
 UCHAR SetupCount;               Count of setup words
 UCHAR Reserved3;                Reserved (pad above to word boundary)
 USHORT Setup[SetupCount];       Setup words (# = SetupWordCount)
```

```
USHORT ByteCount;               Count of data bytes
STRING Name[];                  Must be NULL
UCHAR Pad[];                    Pad to SHORT or LONG
UCHAR Parameters[               Parameter bytes (# = ParameterCount)
   ParameterCount];
UCHAR Pad1[];                   Pad to SHORT or LONG
UCHAR Data[DataCount];          Data bytes (# = DataCount)
```

The interim server response will consist of two fields:

```
UCHAR WordCount;               \\ Count of parameter words = 0
USHORT ByteCount;              \\ Count of data bytes = 0
```

The following list describes the format of the TRANSACTION2 secondary client request:

```
Secondary Client Request          Description
========================          ============
Command                           SMB_COM_TRANSACTION_SECONDARY
UCHAR WordCount;                  Count of parameter words = 8
USHORT TotalParameterCount;       Total parameter bytes being sent
USHORT TotalDataCount;            Total data bytes being sent
USHORT ParameterCount;            Parameter bytes sent this buffer
USHORT ParameterOffset;           Offset (from header start) to Parameters
USHORT ParameterDisplacement;     Displacement of these Parameter bytes
USHORT DataCount;                 Data bytes sent this buffer
USHORT DataOffset;                Offset (from header start) to data
USHORT DataDisplacement;          Displacement of these data bytes
USHORT Fid;                          FID for handle based requests, else
                                     0xFFFF.  This field is present only
                                     if this is an SMB_COM_TRANSACTION2
                                     request.
USHORT ByteCount;                 Count of data bytes
UCHAR Pad[];                      Pad to SHORT or LONG
UCHAR Parameters[                 Parameter bytes (# = ParameterCount)
   ParameterCount];
UCHAR Pad1[];                     Pad to SHORT or LONG
UCHAR Data[DataCount];            Data bytes (# = DataCount)
```

And, the fields of the server response are described in the following list:

```
Server Response                   Description
===============                   ============
UCHAR WordCount;                  Count of data bytes; value = 10 +
                                     SetupCount
USHORT TotalParameterCount;       Total parameter bytes being sent
USHORT TotalDataCount;            Total data bytes being sent
USHORT Reserved;
USHORT ParameterCount;            Parameter bytes sent this buffer
USHORT ParameterOffset;           Offset (from header start) to Parameters
USHORT ParameterDisplacement;     Displacement of these Parameter
                                             bytes
USHORT DataCount;                 Data bytes sent this buffer
USHORT DataOffset;                Offset (from header start) to data
USHORT DataDisplacement;          Displacement of these data bytes
UCHAR SetupCount;                 Count of setup words
```

```
UCHAR Reserved2;                Reserved (pad above to word boundary)
USHORT Setup[SetupWordCount];   Setup words (# = SetupWordCount)
USHORT ByteCount;               Count of data bytes
UCHAR Pad[];                    Pad to SHORT or LONG
UCHAR Parameters[               Parameter bytes (# = ParameterCount)
   ParameterCount];
UCHAR Pad1[];                   Pad to SHORT or LONG
UCHAR Data[DataCount];          Data bytes (# = DataCount)
```

3.15.2. SMB_COM_NT_TRANSACTION Formats

The following list describes the format of the TRANSACTION primary client request:

```
Primary Client Request          Description
======================          ============
UCHAR WordCount;                Count of parameter words; value =
                                         (19 + SetupCount)
UCHAR MaxSetupCount;            Max setup words to return
USHORT Reserved;
ULONG TotalParameterCount;      Total parameter bytes being sent
ULONG TotalDataCount;           Total data bytes being sent
ULONG MaxParameterCount;        Max parameter bytes to return
ULONG MaxDataCount;             Max data bytes to return
ULONG ParameterCount;           Parameter bytes sent this buffer
ULONG ParameterOffset;          Offset (from header start) to Parameters
ULONG DataCount;                Data bytes sent this buffer
ULONG DataOffset;               Offset (from header start) to data
UCHAR SetupCount;               Count of setup words
USHORT Function;                The transaction function code
UCHAR Buffer[1];
USHORT Setup[SetupWordCount];   Setup words
USHORT ByteCount;               Count of data bytes
UCHAR Pad1[];                   Pad to LONG
UCHAR Parameters[               Parameter bytes
   ParameterCount];
UCHAR Pad2[];                   Pad to LONG
UCHAR Data[DataCount];          Data bytes
```

The interim server response will consist of two fields:

```
UCHAR WordCount;    \\ Count of parameter words = 0
USHORT ByteCount;   \\ Count of data bytes = 0
```

The following list describes the format of the TRANSACTION secondary client request:

```
Secondary Client Request        Description
========================        ============
UCHAR WordCount;                Count of parameter words = 18
UCHAR Reserved[3];              MUST BE ZERO
ULONG TotalParameterCount;      Total parameter bytes being sent
ULONG TotalDataCount;           Total data bytes being sent
ULONG ParameterCount;           Parameter bytes sent this buffer
ULONG ParameterOffset;          Offset (from header start) to
                                 Parameters
```

```
ULONG ParameterDisplacement;     Specifies the offset from the start
                                  of the overall parameter block to
                                  the parameter bytes that are
                                  contained in this message
ULONG DataCount;                 Data bytes sent this buffer
ULONG DataOffset;                Offset (from header start) to data
ULONG DataDisplacement;          Specifies the offset from the start
                                  of the overall data block to the
                                  data bytes that are contained in
                                  this message
UCHAR Reserved1;
USHORT ByteCount;                Count of data bytes
UCHAR Pad1[];                    Pad to LONG
UCHAR Parameters[                Parameter bytes
   ParameterCount];
UCHAR Pad2[];                    Pad to LONG
UCHAR Data[DataCount];           Data bytes
```

And, the fields of the server response are described in the following list:

```
Server Response                  Description
===============                  ============
UCHAR WordCount;                 Count of data bytes; value = 18 +
                                  SetupCount
UCHAR Reserved[3];
ULONG TotalParameterCount;       Total parameter bytes being sent
ULONG TotalDataCount;            Total data bytes being sent
ULONG ParameterCount;            Parameter bytes sent this buffer
ULONG ParameterOffset;           Offset (from header start) to
                                  Parameters
ULONG ParameterDisplacement;     Specifies the offset from the start
                                  of the overall parameter block to
                                  the parameter bytes that are
                                  contained in this message
ULONG DataCount;                 Data bytes sent this buffer
ULONG DataOffset;                Offset (from header start) to data
ULONG DataDisplacement;          Specifies the offset from the start
                                  of the overall data block to the
                                  data bytes that are contained in
                                  this message
UCHAR SetupCount;                Count of setup words
USHORT Setup[SetupWordCount];    Setup words
USHORT ByteCount;                Count of data bytes
UCHAR Pad1[];                    Pad to LONG
UCHAR Parameters[                Parameter bytes
   ParameterCount];
UCHAR Pad2[];                    Pad to SHORT or LONG
UCHAR Data[DataCount];           Data bytes
```

3.15.3. Functional Description

The transaction Setup information and/or Parameters define functions specific to a particular resource on a particular server. Therefore the functions supported are not defined by the

transaction sub-protocol. The transaction protocol simply provides a means of delivering them and retrieving the results.

The number of bytes needed in order to perform the transaction request may be more than will fit in a single buffer.

At the time of the request, the client knows the number of parameter and data bytes expected to be sent and passes this information to the server via the primary request (TotalParameterCount and TotalDataCount). This may be reduced by lowering the total number of bytes expected (TotalParameterCount and TotalDataCount) in each (if any) secondary request.

When the amount of parameter bytes received (total of each ParameterCount) equals the total amount of parameter bytes expected (smallest TotalParameterCount) received, then the server has received all the parameter bytes.

Likewise, when the amount of data bytes received (total of each DataCount) equals the total amount of data bytes expected (smallest TotalDataCount) received, then the server has received all the data bytes.

The parameter bytes should normally be sent first followed by the data bytes. However, the server knows where each begins and ends in each buffer by the offset fields (ParameterOffset and DataOffset) and the length fields (ParameterCount and DataCount). The displacement of the bytes (relative to start of each) is also known (ParameterDisplacement and DataDisplacement). Thus the server is able to reassemble the parameter and data bytes should the individual requests be received out of sequence.

If all parameter bytes and data bytes fit into a single buffer, then no interim response is expected and no secondary request is sent.

The client knows the maximum amount of data bytes and parameter bytes which may be returned by the server (from MaxParameterCount and MaxDataCount of the request). Thus the client initializes its bytes expected variables to these values. The server then informs the client of the actual amounts being returned via each message of the server response (TotalParameterCount and TotalDataCount). The server may reduce the expected bytes by lowering the total number of bytes expected (TotalParameterCount and/or TotalDataCount) in each (any) response.

When the amount of parameter bytes received (total of each ParameterCount) equals the total amount of parameter bytes expected (smallest TotalParameterCount) received, then the client has received all the parameter bytes.

Likewise, when the amount of data bytes received (total of each DataCount) equals the total amount of data bytes expected (smallest TotalDataCount) received, then the client has received all the data bytes.

The parameter bytes should normally be returned first followed by the data bytes. However, the client knows where each begins and ends in each buffer by the offset fields (ParameterOffset and DataOffset) and the length fields (ParameterCount and DataCount). The displacement of the bytes (relative to start of each) is also known (ParameterDisplacement and DataDisplacement). The client is able to reassemble the parameter and data bytes should the server responses be received out of sequence.

The flow for these transactions over a connection oriented transport is:

1. The client sends the primary client request identifying the total bytes (both parameters and data) which are expected to be sent and contains the set up words and as many of the parameter and data bytes as will fit in a negotiated size buffer. This request also identifies the maximum number of bytes (setup, parameters and data) the server is to return on the transaction completion. If all the bytes fit in the single buffer, skip to step 4.
2. The server responds with a single interim response meaning "OK, send the remainder of the bytes" or (if error response) terminate the transaction.
3. The client then sends another buffer full of bytes to the server. This step is repeated until all of the bytes are sent and received.
4. The Server sets up and performs the transaction with the information provided.
5. Upon completion of the transaction, the server sends back (up to) the number of parameter and data bytes requested (or as many as will fit in the negotiated buffer size). This step is repeated until all result bytes have been returned.

The flow for the transaction protocol when the request parameters and data do not all fit in a single buffer is:

Client	< -- >	Server
Primary TRANSACTION request	- >	
	< -	Interim Server Response
Secondary TRANSACTION request 1	- >	
Secondary TRANSACTION request 2	- >	
Secondary TRANSACTION request n	- >	
	< -	Transaction response 1
	< -	Transaction response 2
	< -	Transaction response m

The flow for the transaction protocol when the request parameters and data do all fit in a single buffer is:

Client	< -- >	Server
Primary TRANSACTION request	- >	
	< -	Transaction response 1
	< -	Transaction response 2
	< -	Transaction response m

The primary transaction request through the final response make up the complete transaction exchange, thus the Tid, Pid, Uid and Mid must remain constant and can be used as appropriate by both the server and the client. Of course, other SMB requests may intervene as well.

There are (at least) three ways that actual server responses have been observed to differ from what might be expected. First, some servers will send Pad bytes to move the DataOffset to a 2- or 4-byte boundary even if there are no data bytes; the point here is that the ByteCount must be used instead of ParameterOffset plus ParameterCount to infer the actual message length. Second, some servers always return MaxParameterCount bytes even if the particular Transact2 has no parameter response. Finally, in case of an error, some servers send the "traditional WordCount==0/ByteCount==0" response while others generate a Transact response format.

3.15.4. SMB_COM_TRANSACTION Operations

DCE/RPC documents were defined by the Open Group (TOG) used to be called the X/open group. CIFS uses DCE/RPC to process Server and User management information, like logon information, Local Security, Account management, Server/Workstation services and CIFS networking management functions (like browsing and domain controller management). DCE/RPC are implemented on top of SMB. SMB protocol is used as a transport for the DCE/RPC protocol. DCE/RPC uses Protocol Data Unit (PDU) fragments to communicate. The PDUs are totally independent of the SMB transmission size. So PDU can span over multiple SMB transmission boundaries and multiple PDUs can be transmitted in a single SMB transmission. Name Pipe are used as the transmission vehicle. Once and Named Pipe is opened all the DCE/RPC calls related to that Name Pipe will be written and read through SMB_COM_TRANSCATION operation. SMB_COM_TRANSACTION will communicate to the Name Pipe with as much PDU fragments it can contains, the rest of the fragments will follow with either SMBReadX or SMBWriteX. Some of the RPC calls are defined at Appendix E.

The "smb com transaction" style subprotocols are used mostly as MS RPC commands for managing the server and the client. Mail Slots are used for broadcasting and informing the other nodes on the networks. Named Pipes are mostly used for RPC. The details of the use of these RPCs are outside of the scope of this document. The following section describes the data format, but not the content of the content of the RPC. After the client or the server has open a Name Pipe the RPC are communicated using that pipe.

3.15.4.1. Mail Slot Transaction Protocol

The only transaction allowed to a mailslot is a mailslot write. The following table shows the interpretation of parameters for a mailslot transaction:

Name	Value	Description
Command	SMB_COM_TRANSACTION	
Name	\MAILSLOT\<name>	STRING Name of mail slot to write
SetupCount	3	
Setup[0]	1	Command code == write mailslot
Setup[1]		Ignored
Setup[2]		Ignored
TotalDataCount	n	Size of data to write to the mailslot
Data[n]		The data to write to the mailslot

3.15.4.2. Server Announcement Mailslot Transaction

A server announces its presence on the network by periodically transmitting an announcement mailslot message to a well known name. The server initially announces itself every minute, but as the server stays up for longer and longer periods, it should stretch out its announcement period to a maximum of once every 12 minutes. If a server has not been heard from for three announcements, it is considered unavailable. The announcements can be received by any entity on the network wishing to keep a reasonably up to date view of the available network servers.

Systems wishing to be visible on the network and compatible with LANMAN 1.0 periodically send the following announcement:

Name	Value	Description
Command	SMB_COM_TRANSACTION	
Name	\MAILSLOT\LANMAN	
SetupCount	3	
Setup[0]	1	Command code -- write mailslot
Setup[1]		Ignored
Setup[2]		Ignored
TotalDataCount	N	Size of following data to write to the mailslot

Data [n]	Description
USHORT Opcode;	Announcement (value == 1)
ULONG InstalledServices;	Bit mask describing the services running on the system
	0x1 SMB Workstation
	0x2 SMB Server
	0x4 SQL Server
	0x800 UNIX Operating System
	0x1000 NT Operating System
UCHAR MajorVersion;	Major version number of network software
UCHAR MinorVersion;	Minor version number of network software
USHORT Periodicity;	Announcement frequency in seconds
UCHAR ServerName[];	NULL terminated ASCII server name
UCHAR ServerComment[];	NULL terminated ASCII server comment (up to 43 bytes in length)

The NETBIOS address for this mailslot transaction is the domain name padded with blanks and having a zero as the sixteenth octet.

A client can cause LANMAN 1.0 severs to announce themselves to the client by sending the following mailslot transaction to the specific computer of interest or to the domain name as previously described:

Name	Value	Description
Command	SMB_COM_TRANSACTION	
Name	\MAILSLOT\LANMAN	
SetupCount	3	
Setup[0]	1	Command code -- write mailslot
Setup[1]		Ignored
Setup[2]		Ignored
TotalDataCount	N	Size of following data to write to the mailslot

Data [n]	Description
USHORT Opcode;	Request announcement (value == 2)
UCHAR ResponseComputerName[];	NULL terminated ASCII name to which the announcement response should be sent.

Nodes wishing to be visible on the network and compatible with systems using `Windows for Workgroups 3.1a` and later dialects periodically send the following directed mailslot message to a NETBIOS address consisting of the domain name padded with blanks and having a 0x1D in the sixteenth octet.

Name	Value	Description
Command	SMB_COM_TRANSACTION	
Name	\MAILSLOT\LANMAN	
SetupCount	3	
Setup[0]	1	Command code -- write mailslot
Setup[1]		Ignored
Setup[2]		Ignored
TotalDataCount	n	Size of following data to write to the mailslot

Data [n]	Description
UCHAR BrowseType;	Announcement (value == 1)
UCHAR Reserved;	value == 0
ULONG Periodicity;	Announcement frequency in milliseconds
UCHAR ServerName[16]	Name of this node doing the announcement. ServerName[16] == 0
UCHAR VersionMajor;	Major version number of network software
UCHAR VersionMinor;	Minor version number of network software
ULONG InstalledServices;	Bit mask describing the services running on the system
	0x1 SMB Workstation
	0x2 SMB Server
	0x4 SQL Server
	0x800 UNIX Operating System
	0x1000 NT Operating System
ULONG AStrangeValue;	== 0xAA55001F
UCHAR ServerComment[44];	NULL terminated ASCII server comment (up to 44 bytes in length)

3.15.4.3. Named Pipe Transaction Protocol

A named pipe `SMB_COM_TRANSACTION` is used to wait for the specified named pipe to become available (WaitNmPipe) or perform a logical "open -> write -> read -> close" of the pipe (CallNmPipe), along with other functions defined below.

The identifier "\PIPE\<name>" denotes a named pipe transaction, where the <name> is the pipe name to apply the transaction against.

Name	Value	Description
Command	SMB_COM_TRANSACTION	
Name	\PIPE\<name>	Name of pipe for operation
SetupCount	2	
Setup[0]	See Below	Subcommand code
Setup[1]	Fid of pipe	If required
TotalDataCount	n	Size of data
Data[n]		If required

The subcommand codes, placed in *SETUP[0]*, for named pipe operations are:

SubCommand Code	Value	Description
CallNamedPipe	0x54	open/write/read/close pipe
WaitNamedPipe	0x53	wait for pipe to be nonbusy
PeekNmPipe	0x23	read but don't remove data
QNmPHandState	0x21	query pipe handle modes
SetNmPHandState	0x01	set pipe handle modes
QNmPipeInfo	0x22	query pipe attributes
TransactNmPipe	0x26	write/read operation on pipe
RawReadNmPipe	0x11	read pipe in "raw" (non message mode)
RawWriteNmPipe	0x31	write pipe "raw" (non message mode) */

3.15.4.4. CallNamedPipe

This command is used to implement the Win32 CallNamedPipe() API remotely. The CallNamedPipe function connects to a message-type pipe (and waits if an instance of the pipe is not available), writes to and reads from the pipe, and then closes the pipe.

This form of the transaction protocol sends no parameter bytes, thus the bytes to be written to the pipe are sent as data bytes and the bytes read from the pipe are returned as data bytes.

The number of bytes being written is defined by *TOTALDATACOUNT* and the maximum number of bytes to return is defined by *MAXDATACOUNT*.

On the response *TOTALPARAMETERCOUNT* is 0 (no Parameter bytes to return), *TOTALDATACOUNT* indicates the amount of databytes being returned in total and *DATACOUNT* identifies the amount of data being returned in each buffer.

Note that the full form of the Transaction protocol can be used to write and read up to 65,535 bytes each utilizing the secondary requests and responses.

3.15.4.5. WaitNamedPipe

The command is used to implement the Win32 WaitNamedPipe() API remotely. The WaitNamedPipe function waits until either a time-out interval elapses or an instance of the specified named pipe is available to be connected to (that is, the pipe's server process has a pending ConnectNamedPipe operation on the pipe).

The server will wait up to *TIMEOUT* milliseconds for a pipe of the name given to become available. Note that although the timeout is specified in milliseconds, by the time that the timeout occurs and the client receives the timed out response much more time than specified may have occurred.

This form of the transaction protocol sends no data or parameter bytes. The response also contains no data or parameters. If the transaction response indicates success, the pipe may now be available. However, this request does not reserve the pipe, thus all waiting programs may race to get the pipe now available. The losers will get an error on the pipe open attempt.

3.15.4.6. PeekNamedPipe

This form of the pipe Transaction protocol is used to implement the Win32 PeekNamePipe() API remotely. The PeekNamedPipe function copies data from a named or anonymous pipe into a buffer without removing it from the pipe. It also returns information about data in the pipe.

TOTALPARAMETERCOUNT and *TOTALDATACOUNT* should be 0 for this request. The *FID* of the pipe to which this request should be applied is in Setup[1]. *MAXPARAMETERCOUNT* should be set to 6, requesting 3 words of information about the pipe, and *MAXDATACOUNT* should be set to the number of bytes to "peek".

The response contains the following *PARAMETER WORDS*:

Name	Description
Parameters[0, 1]	Total number of bytes available to be read from the pipe
Parameters[2,3]	Total number of bytes remaining in the message at the "head" of the pipe
Parameters[4,5]	Pipe status.
	1 Disconnected by server
	2 Listening
	3 Connection to server is OK
	4 Server end of pipe is closed

The *DATA* portion of the response is the data peeked from the named pipe.

3.15.4.7. GetNamedPipeHandleState

This form of the pipe transaction protocol is used to implement the Win32 GetNamedPipeHandleState() API. The GetNamedPipeHandleState function retrieves information about a specified named pipe. The information returned can vary during the lifetime of an instance of the named pipe.

This request sends no parameters and no data. The *FID* of the pipe to which this request should be applied is in Setup[1]. *MAXPARAMETERCOUNT* should be set to 2 (requesting the 1 word of information about the pipe) and *MAXDATACOUNT* should be 0 (not reading the pipe).

The response returns one parameter of pipe state information interpreted as:

```
Pipe Handle State Bits
       5 4 3 2 1 0 9 8 7 6 5 4 3 2 1 0
       B E * * T T R R |--- Icount --|
where:
    B - Blocking
        0 =>  reads/writes  block  if  no  data available
        1 =>  reads/writes  return  immediately if no data
    E - Endpoint
    0 => client end of pipe
        1 => server end of pipe
    TT - Type of pipe
        00 => pipe is a byte stream pipe
        01 => pipe is a message pipe
    RR - Read Mode
        00 => Read pipe as a byte stream
        01 => Read messages from pipe
    Icount - 8-bit count to control pipe instancing
```

The E (endpoint) bit is 0 because this handle is the client end of a pipe.

3.15.4.8. SetNamedPipeHandleState

This form of the pipe transaction protocol is used to implement the Win32 SetNamedPipeHandleState() API. The SetNamedPipeHandleState function sets the read mode and the blocking mode of the specified named pipe.

This request sends 1 parameter word (*TOTALPARAMETERCOUNT* = 2) which is the pipe state to be set. The *FID* of the pipe to which this request should be applied is in *SETUP[1].*

The response contains no data or parameters.

The interpretation of the input parameter word is:

```
Pipe Handle State Bits
       5 4 3 2 1 0 9 8 7 6 5 4 3 2 1 0
       B * * * * * R R 0 0 0 0 0 0 0 0
where:
    B - Blocking
        0 =>  reads/writes  block  if  no  data available
        1 =>  reads/writes  return  immediately if no data
    RR - Read Mode
        00 => Read pipe as a byte stream
        01 => Read messages from pipe
```

Note that only the read mode (byte or message) and blocking/nonblocking mode of a named pipe can be changed. Some combinations of parameters may be illegal and will be rejected as an error.

3.15.4.9. GetNamedPipeInfo

This form of the pipe transaction protocol is used to implement the Win32 GetNamedPipeInfo() API. The GetNamedPipeInfo function retrieves information about the specified named pipe.

The request sends 1 parameter word (*TOTALPARAMETERCOUNT* = 2) which is the information level requested and must be set to 1. The *FID* of the pipe to which this request should be applied is in *SETUP[1]. MAXDATACOUNT* should be set to the size of the buffer specified by the user in which to receive the pipe information.

Pipe information is returned in the data area of the response, up to the number of bytes specified. The information is returned in the following format:

Name	Size	Description
OutputBufferSize	USHORT	actual size of buffer for outgoing (server) I/O
InputBufferSize	USHORT	actual size of buffer for incoming (client) I/O
MaximumInstances	UCHAR	Maximum allowed number of instances
CurrentInstances	UCHAR	Current number of instances
PipeNameLength	UCHAR	Length of pipe name (including the null)
PipeName	STRING	Name of pipe (NOT including \\NodeName - \\NodeName is prepended to this string by the client before passing back to the user)

3.15.4.10. TransactNamedPipe

This form of the pipe transaction protocol is used to implement the Win32 TransactNamedPipe() API. The TransactNamedPipe function combines into a single network operation the functions that write a message to and read a message from the specified named pipe.

It provides an optimum way to implement transaction-oriented dialogs. TransactNamedPipe will fail if the pipe currently contains any unread data or is not in message read mode. Otherwise the call will write the entire request data bytes to the pipe and then read a response from the pipe and return it in the data bytes area of the response protocol. In the transaction request, *SETUP[1]* must contain the *FID* of the pipe.

If *NAME* is \PIPE\LANMAN, this is a server API request. The request encoding is:

Request Field	Description
Parameters[0->1]	API #
Parameters[2->N]	ASCIIZ RAP description of input structure
Parameters[N->X]	The input structure

The response is formatted as:

Response Field	Description
Parameters[0->1]	Result Status
Parameters[2->3]	Offset to result structure

The state of blocking/nonblocking has no effect on this protocol (TransactNamedPipe does not return until a message has been read into the response protocol). If *MAXDATACOUNT* is too small to contain the response message, an error is returned.

3.15.4.11. RawReadNamedPipe

RawReadNamedPipe reads bytes directly from a pipe, regardless of whether it is a message or byte pipe. For a byte pipe, this is exactly like `SMB_COM_READ`. For a message pipe, this is exactly

like reading the pipe in byte read mode, except message headers will also be returned in the buffer (note that message headers will always be returned in toto--never split at a byte boundary).

This request sends no parameters or data to the server, and *SETUP[1]* must contain the *FID* of the pipe to read. *MAXDATACOUNT* should contain the number of bytes to read raw.

The response will return 0 parameters, and *DATACOUNT* will be set to the number of bytes read.

3.15.4.12.RawWriteNamedPipe

RawWriteNamedPipe puts bytes directly into a pipe, regardless of whether it is a message or byte pipe. The data will include message headers if it is a message pipe. This call ignores the blocking/nonblocking state and always acts in a blocking manner. It returns only after all bytes have been written.

The request sends no parameters. *SETUP[1]* must contain the *FID* of the pipe to write. *TOTALDATACOUNT* is the total amount of data to write to the pipe. Writing zero bytes to a pipe is an error unless the pipe is in message mode.

The response contains no data and one parameter word. If no error is returned, the one parameter word indicates the number of the requested bytes that have been "written raw" to the specified pipe.

3.16. Valid SMB Requests by Negotiated Dialect

CIFS clients and servers may exchange the following SMB messages if the "PC NETWORK PROGRAM 1.0" dialect is negotiated:

SMB_COM_CREATE_DIRECTORY	SMB_COM_DELETE_DIRECTORY
SMB_COM_OPEN	SMB_COM_CREATE
SMB_COM_CLOSE	SMB_COM_FLUSH
SMB_COM_DELETE	SMB_COM_RENAME
SMB_COM_QUERY_INFORMATION	SMB_COM_SET_INFORMATION
SMB_COM_READ	SMB_COM_WRITE
SMB_COM_LOCK_BYTE_RANGE	SMB_COM_UNLOCK_BYTE_RANGE
SMB_COM_CREATE_TEMPORARY	SMB_COM_CREATE_NEW
SMB_COM_CHECK_DIRECTORY	SMB_COM_PROCESS_EXIT
SMB_COM_SEEK	SMB_COM_TREE_CONNECT
SMB_COM_TREE_DISCONNECT	SMB_COM_NEGOTIATE
SMB_COM_QUERY_INFORMATION_DISK	SMB_COM_SEARCH
SMB_COM_OPEN_PRINT_FILE	SMB_COM_WRITE_PRINT_FILE
SMB_COM_CLOSE_PRINT_FILE	SMB_COM_GET_PRINT_QUEUE

If the "LANMAN 1.0" dialect is negotiated, all of the messages in the previous list must be supported. Clients negotiating LANMAN 1.0 and higher dialects will probably no longer send SMB_COM_PROCESS_EXIT, and the response format for SMB_COM_NEGOTIATE is modified as well. New messages introduced with the LANMAN 1.0 dialect are:

SMB_COM_LOCK_AND_READ	SMB_COM_WRITE_AND_UNLOCK
SMB_COM_READ_RAW	SMB_COM_READ_MPX
SMB_COM_WRITE_MPX	SMB_COM_WRITE_RAW
SMB_COM_WRITE_COMPLETE	SMB_COM_WRITE_MPX_SECONDARY
SMB_COM_SET_INFORMATION2	SMB_COM_QUERY_INFORMATION2
SMB_COM_LOCKING_ANDX	SMB_COM_TRANSACTION
SMB_COM_TRANSACTION_SECONDARY	SMB_COM_IOCTL
SMB_COM_IOCTL_SECONDARY	SMB_COM_COPY

```
SMB_COM_MOVE                    SMB_COM_ECHO
SMB_COM_WRITE_AND_CLOSE         SMB_COM_OPEN_ANDX
SMB_COM_READ_ANDX               SMB_COM_WRITE_ANDX
SMB_COM_SESSION_SETUP_ANDX      SMB_COM_TREE_CONNECT_ANDX
SMB_COM_FIND                    SMB_COM_FIND_UNIQUE
SMB_COM_FIND_CLOSE
```

The "LM1.2X002" dialect introduces these new SMBs:

```
SMB_COM_TRANSACTION2            SMB_COM_TRANSACTION2_SECONDARY
SMB_COM_FIND_CLOSE2             SMB_COM_LOGOFF_ANDX
```

"NT LM 0.12" dialect introduces:

```
SMB_COM_NT_TRANSACT             SMB_COM_NT_TRANSACT_SECONDARY
SMB_COM_NT_CREATE_ANDX          SMB_COM_NT_CANCEL
SMB_COM_NT_RENAME
```

Capabilities are used to determine which SMB requests a server supports. However, they do not directly map to which info levels associated with that particular request are supported. In the event that a client sends a request with an info-level that the server does not support or recognize (if it is legacy), it should return STATUS_UNSUPPORTED (or the non-NT equivalent). The extended functionality that was added later is then simply not available to client applications who would ask for it. (If a file system or SMB server does not support unique file ID's, then the query file information asking for it would return Unsupported, where as the query for other types of file information would return successfully.)

4. SMB Requests

This section lists the "best practice" SMB requests -- ones that would permit a client to exercise full CIFS functionality and optimum performance when interoperating with a server speaking the latest dialect as of this writing ("NT LM 0.12").

Note that, as of this writing, no existing client restricts itself to only these requests, so no useful server can be written that supports just them. The classification is provided so that future clients will be written to permit future servers to be simpler.

4.1. Session Requests

4.1.1. NEGOTIATE: Negotiate Protocol

The following list describes the format of the NEGOTIATE client request:

```
Client Request                Description
==============                ============
 UCHAR WordCount;              Count of parameter words = 0
 USHORT ByteCount;             Count of data bytes; min = 2
 struct {
   UCHAR BufferFormat;         0x02 -- Dialect
   UCHAR DialectName[];        ASCII null-terminated string
 } Dialects[];
```

The Client sends a list of dialects with which it can communicate. The response is a selection of one of those dialects (numbered 0 through n) or -1 (hex FFFF) indicating that none of the dialects were acceptable. The negotiate message is binding on the virtual circuit and must be sent. One and only one negotiate message may be sent, subsequent negotiate requests will be rejected with an error response and no action will be taken.

The protocol does not impose any particular structure to the dialect strings. Implementers of particular protocols may choose to include, for example, version numbers in the string.

If the server does not understand any of the dialect strings, or if PC NETWORK PROGRAM 1.0 is the chosen dialect, the response format is:

```
Server Response               Description
================              ============
 UCHAR WordCount;              Count of parameter words = 1
 USHORT DialectIndex;          Index of selected dialect
 USHORT ByteCount;             Count of data bytes = 0
```

If the chosen dialect is greater than core up to and including LANMAN2.1, the protocol response format is:

```
Server Response               Description
================              ============
 UCHAR WordCount;              Count of parameter words = 13
 USHORT DialectIndex;          Index of selected dialect
 USHORT SecurityMode;          Security mode:
                                bit 0: 0 = share, 1 = user
                                bit 1: 1 = use challenge/response
                                authentication
 USHORT MaxBufferSize;         Max transmit buffer size (>= 1024)
```

```
USHORT MaxMpxCount;           Max pending multiplexed requests
USHORT MaxNumberVcs;          Max VCs between client and server
USHORT RawMode;               Raw modes supported:
                               bit 0: 1 = Read Raw supported
                               bit 1: 1 = Write Raw supported
ULONG SessionKey;             Unique token identifying this session
SMB_TIME ServerTime;          Current time at server
SMB_DATE ServerDate;          Current date at server
USHORT ServerTimeZone;        Current time zone at server
USHORT EncryptionKeyLength;   MUST BE ZERO if not LM2.1
                               dialect
USHORT Reserved;              MUST BE ZERO
USHORT ByteCount;             Count of data bytes
UCHAR EncryptionKey[];        The challenge encryption key
STRING PrimaryDomain[];       The server's primary domain
```

MaxBufferSize is the size of the largest message which the client can legitimately send to the server.

If bit0 of the Flags field is set in the negotiate response, this indicates the server supports the obsolescent SMB_COM_LOCK_AND_READ and SMB_COM_WRITE_AND_UNLOCK client requests.

If the SecurityMode field indicates the server is running in user mode, the client must send appropriate SMB_COM_SESSION_SETUP_ANDX requests before the server will allow the client to access resources. If the SecurityMode field indicates the client should use challenge/response authentication, the client should use the authentication mechanism specified in the Section 2.8.

Clients using the "MICROSOFT NETWORKS 1.03" dialect use a different form of raw reads than documented here, and servers are better off setting RawMode in this response to 0 for such sessions.

If the negotiated dialect is "DOS LANMAN2.1" or "LANMAN2.1", then PrimaryDomain string should be included in this response.

If the negotiated dialect is NT LM 0.12, the response format is:

```
Server Response           Description
================          ============
 UCHAR WordCount;         Count of parameter words = 17
 USHORT DialectIndex;     Index of selected dialect
 UCHAR SecurityMode;      Security mode:
                           bit 0: 0 = share, 1 = user
                           bit 1: 1 = encrypt passwords
                           bit 2: 1 = Security Signatures
                             (SMB sequence numbers) enabled
                           bit 3: 1 = Security Signatures
                             (SMB sequence numbers) required
USHORT MaxMpxCount;       Max pending outstanding requests
USHORT MaxNumberVcs;      Max VCs between client and server
ULONG MaxBufferSize;      Max transmit buffer size
ULONG MaxRawSize;         Maximum raw buffer size
ULONG SessionKey;         Unique token identifying this session
ULONG Capabilities;       Server capabilities
```

```
ULONG SystemTimeLow;      System (UTC) time of the server (low)
ULONG SystemTimeHigh;     System (UTC) time of the server (high)
USHORT ServerTimeZone;    Time zone of server (minutes from UTC)
CHAR EncryptionKeyLength; Length of encryption key
USHORT ByteCount;         Count of data bytes
UCHAR EncryptionKey[];    The challenge encryption key;
                          Present only for Non Extended Security i.e.,
                          CAP_EXTENDED_SECURITY is off in the Capabilities
                           field
UCHAR OemDomainName[];    The name of the domain (in OEM chars);
                          Present Only for Non Extended Security i.e.,
                           CAP_EXTENDED_SECURITY is off in the Capabilities
                           field
UCHAR GUID[16];           A globally unique identifier assigned to the
                           server; Present only when
                            CAP_EXTENDED_SECURITY is on in Capabilities field
 UCHAR SecurityBlob[];    Opaque Security Blob associated with the
                           security package if CAP_EXTENDED_SECURITY
                            is on in the Capabilities field; else challenge
                           for CIFS challenge/response authentication
```

In addition to the definitions above, MaxBufferSize is the size of the largest message which the client can legitimately send to the server. If the client is using a connectionless protocol, MaxBufferSize must be set to the smaller of the server's internal buffer size and the amount of data which can be placed in a response packet.

MaxRawSize specifies the maximum message size the server can send or receive for the obsolescent SMB_COM_WRITE_RAW or SMB_COM_READ_RAW requests.

Connectionless clients must set Sid to 0 in the SMB request header.

The Capabilities field allows the server to tell the client what it supports. The client must not ignore any capabilities specified by the server. The bit definitions are:

Capability Name	Encoding	Meaning
CAP_RAW_MODE	0x0001	The server supports SMB_COM_READ_RAW and SMB_COM_WRITE_RAW (obsolescent)
CAP_MPX_MODE	0x0002	The server supports SMB_COM_READ_MPX and SMB_COM_WRITE_MPX (obsolescent)
CAP_UNICODE	0x0004	The server supports UNICODE strings
CAP_LARGE_FILES	0x0008	The server supports large files with 64 bit offsets
CAP_NT_SMBS	0x0010	The server supports the SMBs particular to the NT LM 0.12 dialect. Implies CAP_NT_FIND.
CAP_RPC_REMOTE_APIS	0x0020	The server supports remote admin API requests via DCE RPC
CAP_STATUS32	0x0040	The server can respond with 32 bit status codes in Status.Status
CAP_LEVEL_II_OPLOCKS	0x0080	The server supports level 2 oplocks

Capability Name	Encoding	Meaning
CAP_LOCK_AND_READ	0x0100	The server supports the SMB, SMB_COM_LOCK_AND_READ
CAP_NT_FIND	0x0200	Reserved
CAP_DFS	0x1000	The server is DFS aware
CAP_INFOLEVEL_PASSTHRU	0x2000	The server supports NT information level requests passing through
CAP_LARGE_READX	0x4000	The server supports large SMB_COM_READ_ANDX (up to 64k)
CAP_LARGE_WRITEX	0x8000	The server supports large SMB_COM_WRITE_ANDX (up to 64k)
CAP_UNIX	0x00800000	The server supports CIFS Extensions for UNIX. (See Appendix D for more detail)
CAP_RESERVED	0x02000000	Reserved for future use
CAP_BULK_TRANSFER	0x20000000	The server supports SMB_BULK_READ, SMB_BULK_WRITE (should be 0, no known implementations)
CAP_COMPRESSED_DATA	0x40000000	The server supports compressed data transfer (BULK_TRANSFER capability is required to support compressed data transfer).
CAP_EXTENDED_SECURITY	0x80000000	The server supports extended security exchanges

Undefined bit MUST be set to zero by servers, and MUST be ignored by clients.

Extended security exchanges provide a means of supporting arbitrary authentication protocols within CIFS. Security blobs are opaque to the CIFS protocol; they are messages in some authentication protocol that has been agreed upon by client and server by some out of band mechanism, for which CIFS merely functions as a transport. When CAP_EXTENDED_SECURITY is negotiated, the server includes a first security blob in its response; subsequent security blobs are exchanged in SMB_COM_SESSION_SETUP_ANDX requests and responses until the authentication protocol terminates.

If the negotiated dialect is NT LM 0.12, then the capabilities field of the Negotiate protocol response indicates whether the server supports Unicode. The server is not required to support Unicode. Unicode is supported in Win9x and NT clients. If Unicode is not supported by the server then some localized of these clients may experience unexpected behavior with filenames, resource names and user names.

ASCII defines the values of 128 characters (0x00 through 0x7F). The remaining 128 values (0x80 through 0xFF) are mapped into different DOS Code Pages (aka the OEM character set). Different localized clients may use different code pages. (For example, Code Page 437 is the default in English based systems). Clients can create file and folder names in their default code page that follows the file naming rules and may contain both ASCII and non-ASCII characters.

4.1.1.1. Errors

```
SUCCESS/SUCCESS
ERRSRV/ERRerror
```

4.1.2. SESSION_SETUP_ANDX: Session Setup

This SMB is used to further "Set up" the session normally just established via the negotiate protocol.

One primary function is to perform a "user logon" in the case where the server is in user level security mode. The Uid in the SMB header is set by the client to be the userid desired for the AccountName and validated by the AccountPassword.

4.1.2.1. Pre NT LM 0.12

If the negotiated protocol is prior to NT LM 0.12, the format of SMB_COM_SESSION_SETUP_ANDX is:

```
Client Request                Description
==============                ============
 UCHAR WordCount;             Count of parameter words = 10
 UCHAR AndXCommand;           Secondary (X) command; 0xFF = none
 UCHAR AndXReserved;          Reserved (must be 0)
 USHORT AndXOffset;           Offset to next command WordCount
 USHORT MaxBufferSize;        Client maximum buffer size
 USHORT MaxMpxCount;          Actual maximum multiplexed pending requests
 USHORT VcNumber;             0 = first (only), nonzero=additional
                                        VC number
 ULONG SessionKey;            Session key (valid iff VcNumber != 0)
 USHORT PasswordLength;       Account password size
 ULONG Reserved;              Must be 0
 USHORT ByteCount;            Count of data bytes; min = 0
 UCHAR AccountPassword[];     Account Password
 STRING AccountName[];        Account Name
 STRING PrimaryDomain[];      Client's primary domain
 STRING NativeOS[];           Client's native operating system
 STRING NativeLanMan[];       Client's native LAN Manager type
```

The server response is:

```
Server Response               Description
===============               ============
 UCHAR WordCount;             Count of parameter words = 3
 UCHAR AndXCommand;           Secondary (X) command;  0xFF =
                                 none
 UCHAR AndXReserved;          Reserved (must be 0)
 USHORT AndXOffset;           Offset to next command WordCount
 USHORT Action;               Request mode:
                               bit0 = logged in as GUEST
 USHORT ByteCount;            Count of data bytes
 STRING NativeOS[];           Server's native operating system
 STRING NativeLanMan[];       Server's native LAN Manager type
 STRING PrimaryDomain[];      Server's primary domain
```

If the server is in "share level security mode", the account name and password should be ignored by the server.

If challenge/response authentication is not being used, AccountPassword should be a null terminated ASCII string with PasswordLength set to the string size including the null; the password will be case insensitive. If challenge/response authentication is being used, then AccountPassword will be the response to the server's challenge, and PasswordLength should be set to its length.

The server validates the name and password supplied and if valid, it registers the user identifier on this session as representing the specified AccountName. The Uid field in the SMB header will then be used to validate access on subsequent SMB requests. The SMB requests where permission checks are required are those which refer to a symbolically named resource such as SMB_COM_OPEN, SMB_COM_RENAME, SMB_COM_DELETE, etc. The value of the Uid is relative to a specific client/server session so it is possible to have the same Uid value represent two different users on two different sessions at the server.

Multiple session setup commands may be sent to register additional users on this session. If the server receives an additional SMB_COM_SESSION_SETUP_ANDX, only the Uid, AccountName and AccountPassword fields need contain valid values (the server MUST ignore the other fields).

The client writes the name of its domain in PrimaryDomain if it knows what the domain name is. If the domain name is unknown, the client either encodes it as a NULL string, or as a question mark.

If bit0 of Action is set, this informs the client that although the server did not recognize the AccountName, it logged the user in as a guest. This is optional behavior by the server, and in any case one would ordinarily expect guest privileges to limited.

Another function of the Session Set Up protocol is to inform the server of the maximum values which will be utilized by this client. Here MaxBufferSize is the maximum message size which the client can receive. Thus although the server may support 16k buffers (as returned in the SMB_COM_NEGOTIATE response), if the client only has 4k buffers, the value of MaxBufferSize here would be 4096. The minimum allowable value for MaxBufferSize is 1024. The SMB_COM_NEGOTIATE response includes the server buffer size supported. Thus this is the maximum SMB message size which the client can send to the server. This size may be larger than the size returned to the server from the client via the SMB_COM_SESSION_SETUP_ANDX protocol which is the maximum SMB message size which the server may send to the client. Thus if the server's buffer size were 4k and the client's buffer size were only 2K, the client could send up to 4k (standard) write requests but must only request up to 2k for (standard) read requests.

The VcNumber field specifies whether the client wants this to be the first VC or an additional VC.

The values for MaxBufferSize, MaxMpxCount, and VcNumber must be less than or equal to the maximum values supported by the server as returned in the SMB_COM_NEGOTIATE response.

If the server gets a SMB_COM_SESSION_SETUP_ANDX request with VcNumber of 0 and other VCs are still connected to that client, they will be aborted thus freeing any resources held by the server. This condition could occur if the client was rebooted and reconnected to the server before the transport level had informed the server of the previous VC termination.

4.1.2.2. NT LM 0.12

If the negotiated SMB dialect is "NT LM 0.12" and the server supports ExtendedSecurity i.e. the CAP_EXTENDED_SECURITY flag is set in the Capabilities field of the Negotiate Response SMB, the Extended Security SessionSetup SMB format is:

```
Client Request                Description
==============                ============
```

```
 UCHAR WordCount;              Count of parameter words = 12
 UCHAR AndXCommand;            Secondary (X) command;  0xFF = none
 UCHAR AndXReserved;           Reserved (must be 0)
 USHORT AndXOffset;            Offset to next command WordCount
 USHORT MaxBufferSize;         Client's maximum buffer size
 USHORT MaxMpxCount;           Actual maximum multiplexed pending
                                requests
 USHORT VcNumber;              0 = first (only), nonzero=additional
                                VC number
 ULONG SessionKey;             Session key (valid iff VcNumber != 0)
 USHORT SecurityBlobLength;    Length of opaque security blob
 ULONG Reserved;               Must be 0
 ULONG Capabilities;           Client capabilities
 USHORT ByteCount;             Count of data bytes; min = 0
 UCHAR SecurityBlob[];         The opaque security blob
 STRING NativeOS[];            Client's native operating system,
                                Unicode
 STRING NativeLanMan[];        Client's native LAN Manager type,
                                Unicode
```

And the server response is:

```
Server Response               Description
================              ============
 UCHAR WordCount;              Count of parameter words = 4
 UCHAR AndXCommand;            Secondary (X) command;  0xFF =
                                none
 UCHAR AndXReserved;           Reserved (must be 0)
 USHORT AndXOffset;            Offset to next command WordCount
 USHORT Action;                Request mode:
                                bit0 = logged in as GUEST
 USHORT SecurityBlobLength;    Length of Security Blob that
                                follows in a later field
 USHORT ByteCount;             Count of data bytes
 UCHAR SecurityBlob[];         SecurityBlob of length specified
                                by the field, SecurityBlobLength
 STRING NativeOS[];            Server's native operating system
 STRING NativeLanMan[];        Server's native LAN Manager type
 STRING PrimaryDomain[];       Server's primary domain
```

There may be multiple round trips involved in the security blob exchange. In that case, the server may return an error STATUS_MORE_PROCESSING_REQUIRED (a value of 0xC0000016) in the SMB status. The client can then repeat the SessionSetupAndX SMB with the rest of the security blob.

If the negotiated SMB dialect is "NT LM 0.12" or later and the server does not support Extended Security (i.e. the CAP_EXTENDED_SECURITY flag in the Capabilities field of the Negotiate Response SMB is not set), the format of the response SMB is unchanged, but the request is:

```
Client Request                Description
==============                ============
 UCHAR WordCount;              Count of parameter words = 13
 UCHAR AndXCommand;            Secondary (X) command;  0xFF = none
 UCHAR AndXReserved;           Reserved (must be 0)
```

```
USHORT AndXOffset;              Offset to next command WordCount
USHORT MaxBufferSize;           Client's maximum buffer size
USHORT MaxMpxCount;             Actual maximum multiplexed pending
                                 requests
USHORT VcNumber;                0 = first (only), nonzero=additional
                                 VC number
ULONG SessionKey;               Session key (valid iff VcNumber != 0)
USHORT                          Account password size, ANSI
   CaseInsensitivePasswordLength;
USHORT                          Account password size, Unicode
   CaseSensitivePasswordLength;
ULONG Reserved;                 Must be 0
ULONG Capabilities;             Client capabilities
USHORT ByteCount;               Count of data bytes; min = 0
UCHAR                           Account Password, ANSI
   CaseInsensitivePassword[];
UCHAR                           Account Password, Unicode
   CaseSensitivePassword[];
UCHAR Reserved2                 Present if Unicode negotiated to even byte
                                boundary
STRING AccountName[];           Account Name, Unicode
STRING PrimaryDomain[];         Client's primary domain, Unicode
STRING NativeOS[];              Client's native operating system,
                                 Unicode
STRING NativeLanMan[];          Client's native LAN Manager type,
                                 Unicode
```

The client expresses its capabilities to the server encoded in the Capabilities field. The format of that field is:

Capability Name	Encoding	Meaning
CAP_UNICODE	0x0004	The client can use UNICODE strings
CAP_LARGE_FILES	0x0008	The client can deal with files having 64 bit offsets
CAP_NT_SMBS	0x0010	The client understands the SMBs introduced with the NT LM 0.12 dialect. Implies CAP_NT_FIND.
CAP_STATUS32	0x0040	The client can receive 32 bit errors encoded in Status.Status
CAP_LEVEL_II_OPLOCKS	0x0080	The client understands Level II oplocks
CAP_NT_FIND	0x0200	Reserved

The entire message sent and received including the optional ANDX SMB must fit in the negotiated maximum transfer size. The following are the only valid SMB commands for AndXCommand for SMB_COM_SESSION_SETUP_ANDX:

```
SMB_COM_TREE_CONNECT_ANDX       SMB_COM_OPEN
SMB_COM_OPEN_ANDX               SMB_COM_CREATE
SMB_COM_CREATE_NEW              SMB_COM_CREATE_DIRECTORY
SMB_COM_DELETE                  SMB_COM_DELETE_DIRECTORY
SMB_COM_FIND                    SMB_COM_FIND_UNIQUE
SMB_COM_COPY                    SMB_COM_RENAME
SMB_COM_NT_RENAME               SMB_COM_CHECK_DIRECTORY
SMB_COM_QUERY_INFORMATION       SMB_COM_SET_INFORMATION
SMB_COM_NO_ANDX_COMMAND         SMB_COM_OPEN_PRINT_FILE
```

```
SMB_COM_GET_PRINT_QUEUE         SMB_COM_TRANSACTION
```

4.1.2.3. Errors

```
ERRSRV/ERRerror - No NEG_PROT issued
ERRSRV/ERRbadpw - Password not correct for given username
ERRSRV/ERRtoomanyuids - Maximum number of users per session exceeded
ERRSRV/ERRnosupport - Chaining of this request to the previous is not
supported
```

4.1.3. LOGOFF_ANDX: User Logoff

This SMB is the inverse of SMB_COM_SESSION_SETUP_ANDX.

```
Client Request                  Description
==============                  ============
 UCHAR WordCount;                Count of parameter words = 2
 UCHAR AndXCommand;              Secondary (X) command;  0xFF =
                                  none
 UCHAR AndXReserved;             Reserved (must be 0)
 USHORT AndXOffset;              Offset to next command WordCount
 USHORT ByteCount;               Count of data bytes = 0
```

The server response is:

```
Server Response                 Description
================                ============
 UCHAR WordCount;                Count of parameter words = 2
 UCHAR AndXCommand;              Secondary (X) command;  0xFF =
                                  none
 UCHAR AndXReserved;             Reserved (must be 0)
 USHORT AndXOffset;              Offset to next command WordCount
 USHORT ByteCount;               Count of data bytes = 0
```

The user represented by Uid in the SMB header is logged off. The server closes all files currently open by this user, and invalidates any outstanding requests with this Uid.

SMB_COM_SESSION_SETUP_ANDX is the only valid AndXCommand for this SMB.

4.1.3.1. Errors

```
ERRSRV/invnid - TID was invalid
ERRSRV/baduid - UID was invalid
```

4.1.4. TREE_CONNECT_ANDX: Tree Connect

The TREE_CONNECT_ANDX client request is defined below:

```
Client Request                  Description
==============                  ============
 UCHAR WordCount;                Count of parameter words = 4
 UCHAR AndXCommand;              Secondary (X) command; 0xFF = none
 UCHAR AndXReserved;             Reserved (must be 0)
 USHORT AndXOffset;              Offset to next command WordCount
 USHORT Flags;                   Additional information
                                  bit 0 set = Disconnect Tid
```

```
USHORT PasswordLength;          Length of Password[]
USHORT ByteCount;               Count of data bytes; min = 3
UCHAR Password[];               Password
STRING Path[];                  Server name and share name
STRING Service[];               Service name
```

The serving machine verifies the combination and returns an error code or an identifier. The full name is included in this request message and the identifier identifying the connection is returned in the Tid field of the SMB header. The Tid field in the client request is ignored. The meaning of this identifier (Tid) is server specific; the client must not associate any standard meaning to it.

If the negotiated dialect is LANMAN1.0 or later, then it is a protocol violation for the client to send this message prior to a successful SMB_COM_SESSION_SETUP_ANDX, and the server ignores Password.

If the negotiated dialect is prior to LANMAN1.0 and the client has not sent a successful SMB_COM_SESSION_SETUP_ANDX request when the tree connect arrives, a user level security mode server must nevertheless validate the client's credentials as discussed earlier in this document.

Path follows UNC style syntax, that is to say it is encoded as \\server\share and it indicates the name of the resource to which the client wishes to connect.

Because Password may be an authentication response, it is a variable length field with the length specified by PasswordLength. If authentication is not being used, Password should be a null terminated ASCII string with PasswordLength set to the string size including the terminating null.

The server can enforce whatever policy it desires to govern share access. Typically, if the server is paused, administrative privilege is required to connect to any share; if the server is not paused, administrative privilege is required only for administrative shares (C$, etc.). Other such policies may include valid times of day, software usage license limits, number of simultaneous server users or share users, etc.

The Service component indicates the type of resource the client intends to access. Valid values are:

Service	Description	Earliest Dialect Allowed
A:	Disk share	PC NETWORK PROGRAM 1.0
LPT1:	Printer	PC NETWORK PROGRAM 1.0
IPC	Named pipe	MICROSOFT NETWORKS 3.0
COMM	Communications device	MICROSOFT NETWORKS 3.0
?????	Any type of device	MICROSOFT NETWORKS 3.0

If bit0 of Flags is set, the tree connection to Tid in the SMB header should be disconnected. If this tree disconnect fails, the error should be ignored.

If the negotiated dialect is earlier than DOS LANMAN2.1, the response to this SMB is:

```
Server Response                 Description
===============                 ============
 UCHAR WordCount;                Count of parameter words = 2
 UCHAR AndXCommand;              Secondary (X) command;  0xFF = none
 UCHAR AndXReserved;             Reserved (must be 0)
 USHORT AndXOffset;              Offset to next command WordCount
```

```
USHORT ByteCount;              Count of data bytes; min = 3
```

If the negotiated is DOS LANMAN2.1 or later, the response to this SMB is:

```
Server Response                Description
================               ============
 UCHAR WordCount;               Count of parameter words = 3
 UCHAR AndXCommand;             Secondary (X) command;  0xFF = none
 UCHAR AndXReserved;            Reserved (must be 0)
 USHORT AndXOffset;             Offset to next command WordCount
 USHORT OptionalSupport;        Optional support bits
                                  SMB_SUPPORT_SEARCH_BITS = 0x0001
                                 Exclusive search bits
                                   ("MUST HAVE BITS") supported
                                   SMB_SHARE_IS_IN_DFS = 0x0002
 USHORT ByteCount;              Count of data bytes; min = 3
 UCHAR Service[];               Service type connected (Always ANSII)
 STRING NativeFileSystem[];     Native file system for this tree
```

NativeFileSystem is the name of the filesystem. Expected values include FAT, NTFS, etc.

Some servers negotiate "DOS LANMAN2.1" dialect or later and still send the "downlevel" (i.e. wordcount==2) response. Valid AndX following commands are:

```
SMB_COM_OPEN              SMB_COM_OPEN_ANDX            SMB_COM_CREATE
SMB_COM_CREATE_NEW        SMB_COM_CREATE_DIRECTORY     SMB_COM_DELETE
SMB_COM_DELETE_DIRECTORY  SMB_COM_FIND                 SMB_COM_COPY
SMB_COM_FIND_UNIQUE       SMB_COM_RENAME
SMB_COM_CHECK_DIRECTORY   SMB_COM_QUERY_INFORMATION
SMB_COM_GET_PRINT_QUEUE   SMB_COM_OPEN_PRINT_FILE
SMB_COM_TRANSACTION       SMB_COM_NO_ANDX_CMD
SMB_COM_SET_INFORMATION   SMB_COM_NT_RENAME
```

4.1.4.1. Errors

```
ERRDOS/ERRnomem
ERRDOS/ERRbadpath
ERRDOS/ERRinvdevice
ERRSRV/ERRaccess
ERRSRV/ERRbadpw
ERRSRV/ERRinvnetname
```

4.1.5. TREE_DISCONNECT: Tree Disconnect

This message informs the server that the client no longer wishes to access the resource connected via a prior SMB_COM_TREE_CONNECT or SMB_COM_TREE_CONNECT_ANDX.

```
Client Request                 Description
===============                ============
 UCHAR WordCount;               Count of parameter words = 0
 USHORT ByteCount;              Count of data bytes = 0
```

The resource sharing connection identified by Tid in the SMB header is logically disconnected from the server. Tid is invalidated; it will not be recognized if used by the client for subsequent requests. All locks, open files, etc. created on behalf of Tid are released.

```
Server Response                 Description
================                ============
 UCHAR WordCount;                Count of parameter words = 0
 USHORT ByteCount;               Count of data bytes = 0
```

4.1.5.1. Errors

```
ERRSRV/ERRinvnid
ERRSRV/ERRbaduid
```

4.1.6. TRANS2_QUERY_FS_INFORMATION: Get File System Information

This transaction requests information about a filesystem on the server. Its format is:

```
Client Request      Value
================    ======
WordCount           15
TotalParameterCount 2 or 4
MaxSetupCount       0
SetupCount          1 or 2
Setup[0]            TRANS2_QUERY_FS_INFORMATION
```

The request's parameter block encodes InformationLevel (a USHORT), describing the level of filesystem info that should be returned. Values for InformationLevel are specified in the table below.

The filesystem is identified by Tid in the SMB header.

MaxDataCount in the transaction request must be large enough to accommodate the response.

The encoding of the response parameter block depends on the InformationLevel requested. Information levels whose values are greater than 0x102 are mapped to corresponding operating system calls (NtQueryVolumeInformationFile calls) by the server. The two levels below 0x102 are described below. The requested information is placed in the Data portion of the transaction response.

Information Level	Value
SMB_INFO_ALLOCATION	1
SMB_INFO_VOLUME	2
SMB_QUERY_FS_VOLUME_INFO	0x102
SMB_QUERY_FS_SIZE_INFO	0x103
SMB_QUERY_FS_DEVICE_INFO	0x104
SMB_QUERY_FS_ATTRIBUTE_INFO	0x105
SMB_QUERY_CIFS_UNIX_INFO	0x200
SMB_QUERY_MAC_FS_INFO	0x301

The following sections describe the InformationLevel dependent encoding of the data part of the transaction response.

4.1.6.1. SMB_INFO_ALLOCATION

```
InformationLevel
```

```
Data Block Encoding   Description
====================  ============
ULONG idFileSystem;    File system identifier (NT server always returns 0)
ULONG cSectorUnit;     Number of sectors per allocation unit
ULONG cUnit;           Total number of allocation units
ULONG cUnitAvail;      Total number of available allocation units
USHORT cbSector;       Number of bytes per sector
```

4.1.6.2. SMB_INFO_VOLUME

```
InformationLevel
Data Block Encoding    Description
====================   ============
ULONG ulVsn;           Volume serial number
UCHAR cch;             Number of  characters in Label
STRING Label;          The volume label
```

4.1.6.3. SMB_QUERY_FS_VOLUME_INFO

```
InformationLevel
Data Block Encoding   Description
====================  ============
SMB_TIME               Volume Creation Time
ULONG                  Volume Serial Number
ULONG                  Length of Volume Label in bytes
BYTE                   Reserved
BYTE                   Reserved
STRING Label;          The volume label
```

4.1.6.4. SMB_QUERY_FS_SIZE_INFO

```
InformationLevel
Data Block Encoding    Description
====================   ============
LARGE_INTEGER          Total Number of Allocation units on the Volume
LARGE_INTEGER          Number of free Allocation units on the Volume
ULONG                  Number of sectors in each Allocation unit
ULONG                  Number of bytes in each sector
```

4.1.6.5. SMB_QUERY_FS_DEVICE_INFO

```
InformationLevel
Data Block Encoding    Description
====================   ======
ULONG                  DeviceType; Values as specified below
ULONG                  Characteristics of the device; Values as specified
below
```

For DeviceType, note that the values 0-32767 are reserved for the exclusive use of Microsoft Corporation. The following device types are currently defined:

```
FILE_DEVICE_BEEP                0x00000001
FILE_DEVICE_CD_ROM              0x00000002
FILE_DEVICE_CD_ROM_FILE_SYSTEM  0x00000003
```

```
FILE_DEVICE_CONTROLLER          0x00000004
FILE_DEVICE_DATALINK            0x00000005
FILE_DEVICE_DFS                 0x00000006
FILE_DEVICE_DISK                0x00000007
FILE_DEVICE_DISK_FILE_SYSTEM    0x00000008
FILE_DEVICE_FILE_SYSTEM         0x00000009
FILE_DEVICE_INPORT_PORT         0x0000000a
FILE_DEVICE_KEYBOARD            0x0000000b
FILE_DEVICE_MAILSLOT            0x0000000c
FILE_DEVICE_MIDI_IN             0x0000000d
FILE_DEVICE_MIDI_OUT            0x0000000e
FILE_DEVICE_MOUSE               0x0000000f
FILE_DEVICE_MULTI_UNC_PROVIDER  0x00000010
FILE_DEVICE_NAMED_PIPE          0x00000011
FILE_DEVICE_NETWORK             0x00000012
FILE_DEVICE_NETWORK_BROWSER     0x00000013
FILE_DEVICE_NETWORK_FILE_SYSTEM 0x00000014
FILE_DEVICE_NULL                0x00000015
FILE_DEVICE_PARALLEL_PORT       0x00000016
FILE_DEVICE_PHYSICAL_NETCARD    0x00000017
FILE_DEVICE_PRINTER             0x00000018
FILE_DEVICE_SCANNER             0x00000019
FILE_DEVICE_SERIAL_MOUSE_PORT   0x0000001a
FILE_DEVICE_SERIAL_PORT         0x0000001b
FILE_DEVICE_SCREEN              0x0000001c
FILE_DEVICE_SOUND               0x0000001d
FILE_DEVICE_STREAMS             0x0000001e
FILE_DEVICE_TAPE                0x0000001f
FILE_DEVICE_TAPE_FILE_SYSTEM    0x00000020
FILE_DEVICE_TRANSPORT           0x00000021
FILE_DEVICE_UNKNOWN             0x00000022
FILE_DEVICE_VIDEO               0x00000023
FILE_DEVICE_VIRTUAL_DISK        0x00000024
FILE_DEVICE_WAVE_IN             0x00000025
FILE_DEVICE_WAVE_OUT            0x00000026
FILE_DEVICE_8042_PORT           0x00000027
FILE_DEVICE_NETWORK_REDIRECTOR  0x00000028
FILE_DEVICE_BATTERY             0x00000029
FILE_DEVICE_BUS_EXTENDER        0x0000002a
FILE_DEVICE_MODEM               0x0000002b
FILE_DEVICE_VDM                 0x0000002c
```

Some of these device types are not currently accessible over the network, and may never be accessible on the network. Some may change to be accessible in the future. The values for device types that will never be accessible over the network may be redefined to be “reserved”.

For the encoding of “Characteristics” in the protocol request, this field is the sum of any of the following:

```
FILE_REMOVABLE_MEDIA            0x00000001
FILE_READ_ONLY_DEVICE           0x00000002
FILE_FLOPPY_DISKETTE            0x00000004
FILE_WRITE_ONE_MEDIA            0x00000008
FILE_REMOTE_DEVICE              0x00000010
```

```
FILE_DEVICE_IS_MOUNTED          0x00000020
FILE_VIRTUAL_VOLUME             0x00000040
```

4.1.6.6. SMB_QUERY_FS_ATTRIBUTE_INFO

```
InformationLevel
Data Block Encoding    Description
====================   ============
ULONG                  File System Attributes;
                        possible values described below
LONG                   Maximum length of each file name component
                        in number of bytes
ULONG                  Length, in bytes, of the name of the file system
STRING                 Name of the file system
```

Where FileSystemAttributes are the sum of any of the following:

```
FILE_CASE_SENSITIVE_SEARCH   0x00000001
FILE_CASE_PRESERVED_NAMES    0x00000002
FILE_PERSISTENT_ACLS         0x00000004
FILE_FILE_COMPRESSION        0x00000008
FILE_VOLUME_QUOTAS           0x00000010
FILE_DEVICE_IS_MOUNTED       0x00000020
FILE_VOLUME_IS_COMPRESSED    0x00008000
```

4.1.6.7. SMB_QUERY_CIFS_UNIX_INFO

```
InformationLevel
Data Block Encoding             Description
====================            ============
UNIT16 MajorVersionNumber;      Major version of CIFS UNIX supported by
                                 server
UNIT16 MinorVersionNumber;      Minor version of CIFS UNIX supported by
                                 server
LARGE_INTEGER Capability;       Capabilities of CIFS UNIX support by
                                 Server
```

Where Capability is the sum of the following:

CIFS_UNIX_FCNTL_CAP	0x1	Reserved. Should be zero
CIFS_UNIX_POSIX_ACL_CAP	0x2	Reserved. Should be zero

4.1.6.8. SMB_QUERY_MAC_FS_INFO

```
InformationLevel
Data Block Encoding    Description
====================   ============
 LARGE_INTEGER         CreationTime; Volume creation time - NT TIME.
 LARGE_INTEGER         ModifyTime; Volume Modify time - NT TIME.
 LARGE_INTEGER         BackUpTime; Volume was last Backup time - NT TIME.
                       Defaults to Create Time.
 ULONG                 NmAlBlks; The number of allocation blocks in the
                       volume
 ULONG                 AlBlkSiz; The allocation block size (in bytes) Must
                       be in multiple of 512 bytes
 ULONG                 FreeBks; The number of unused allocations blocks on
                       the volume
 UCHAR [32];           FndrInfo[32]; Information used by the finder that is
                       always in Big Endian.
                           Bytes 0-3 File Type
                               If a file default to 'TEXT' otherwise
                       default to zero
                           Bytes 4-7 File Creator
                               If a file default to 'dosa' otherwise
                       default to zero
                           Bytes 8-9 a UWORD flags field
                               If hidden item set this UWORD to 0x4000
                       else defaults to zero
                           All other bytes should default to zero and are
                       only changeable by the Macintosh
 LONG                  NmFls; The number of files in the root directory;
                       Zero if not known
 LONG                  NmRtDirs; The number of directories in the root
                       directory; Zero if not known
 LONG                  FilCnt; The number of files on the volume; Zero if
                       not known
 LONG                  DirCnt; The number of directories on the volume;
                       Zero if not known
 LONG                  MacSupportFlags; Must be zero unless you support the
                       other Macintosh options
```

Where *MacSupportFlags* is the sum of any of the following:

SUPPORT_MAC_ACCESS_CNTRL	0x00000010	The server will return folder access control in the Trans2_Find_First2 and Trans2_Find_Next2 message described later in this document.
SUPPORT_MAC_GETSETCOMMENTS	0x00000020	Not currently supported.
SUPPORT_MAC_DESKTOPDB_CALLS	0x00000040	The Server supports setting and getting Macintosh desktop database information using the mechanism in this document.

SUPPORT_MAC_UNIQUE_IDS	0x00000080	The server will return a unique id for files and directories in the Trans2_Find_First2 and Trans2_Find_Next2 message described later in this document.
NO_STREAMS_OR_MAC_SUPPORT	0x00000100	The server will return this flag telling the client that the server does not support streams or the Macintosh extensions. The client will ignore the rest of this message.

4.1.6.9. Errors

```
ERRSRV/invnid - TID was invalid
ERRSRV/baduid - UID was invalid
ERRHRD/ERRnotready - The file system has been removed
ERRHRD/ERRdata - Disk I/O error
ERRSRV/ERRaccess - User does not have rights to perform this operation
ERRSRV/ERRinvdevice - Resource identified by TID is not a file system
```

4.1.7. ECHO: Ping the Server

This request is used to test the connection to the server, and to see if the server is still responding. The client request is defined as:

```
Client Request                  Description
==============                  ============
 UCHAR WordCount;                Count of parameter words = 1
 USHORT EchoCount;               Number of times to echo data back
 USHORT ByteCount;               Count of data bytes; min = 1
 UCHAR Buffer[1];                Data to echo
```

And, the server response is:

```
Server Response                 Description
================                ============
 UCHAR WordCount;                Count of parameter words = 1
 USHORT SequenceNumber;          Sequence number of this echo
 USHORT ByteCount;               Count of data bytes; min = 4
 UCHAR Buffer[1];                Echoed data
```

Each response echoes the data sent, though ByteCount may indicate “no data”. If EchoCount is zero, no response is sent.

Tid in the SMB header is ignored, so this request may be sent to the server even if there are no valid tree connections to the server.

The flow for the ECHO protocol is:

Client Request	< -- >	Server Response
Echo request (EchoCount == n)	- >	
	< -	Echo response 1
	< -	Echo response 2
	< -	Echo response n

4.1.7.1. Errors

```
ERRSRV/ERRbaduid    - UID was invalid
ERRSRV/ERRnoaccess  - session has not been established
ERRSRV/ERRnosupport - ECHO function is not supported
```

4.1.8. NT_CANCEL: Cancel request

This SMB allows a client to cancel a request currently pending at the server. The client request is defined as:

```
Client Request                  Description
==============                  ============
 UCHAR WordCount;                No words are sent (== 0)
 USHORT ByteCount;               No bytes (==0)
```

The Sid, Uid, Pid, Tid, and Mid fields of the SMB are used to locate an pending server request from this session. If a pending request is found, it is "hurried along" which may result in success or failure of the original request. No other response is generated for this SMB.

4.2. File Requests

4.2.1. NT_CREATE_ANDX: Create or Open File

This command is used to create or open a file or a directory. The client request is defined as:

```
Client Request                  Description
==============                  ============
UCHAR WordCount;                Count of parameter words = 24
UCHAR AndXCommand;              Secondary command; 0xFF = None
UCHAR AndXReserved;             Reserved (must be 0)
USHORT AndXOffset;              Offset to next command WordCount
UCHAR Reserved;                 Reserved (must be 0)
USHORT NameLength;              Length of Name[] in bytes
ULONG Flags;                    Create bit set:
                                     0x02 - Request an oplock
                                     0x04 - Request a batch oplock
                                     0x08 - Target of open must be directory
ULONG RootDirectoryFid;         If non-zero, open is relative to
                                     this directory
ACCESS_MASK DesiredAccess;      Access desired (See Section 3.8 for an
                                     explanation of this field)
LARGE_INTEGER AllocationSize;   Initial allocation size
ULONG ExtFileAttributes;        File attributes
ULONG ShareAccess;              Type of share access
ULONG CreateDisposition;        Action if file does/does not exist
```

```
ULONG CreateOptions;            Options to use if creating a file
ULONG ImpersonationLevel;       Security QOS information
UCHAR SecurityFlags;            Security tracking mode flags:
                                     0x1 - SECURITY_CONTEXT_TRACKING
                                     0x2 - SECURITY_EFFECTIVE_ONLY
USHORT ByteCount;               Length of byte parameters
STRING Name[];                  File to open or create
```

The Name parameter contains the full path from the tree connect point unless the RootDirectoryFid is used. To use the RootDirectoryFid perform a NT_CREATE_ANDX to open the directory and then use the returned Fid for subsequent NT_CREATE_ANDX calls to open/create files within that directory.

The DesiredAccess parameter is specified in section 3.8, Access Mask Encoding. If no value is specified, an application can still query attributes without actually accessing the file.

The ExtFileAttributes parameter specifies the file attributes and flags for the file. The parameter's value is the sum of allowed attributes and flags defined in section 3.12, Extended File Attribute Encoding.

The ShareAccess field specifies how the file can be shared. This parameter must be some combination of the following values:

Name	Value	Meaning
FILE_NO_SHARE	0x00000000	Prevents the file from being shared.
FILE_SHARE_READ	0x00000001	Other open operations can be performed on the file for read access.
FILE_SHARE_WRITE	0x00000002	Other open operations can be performed on the file for write access.
FILE_SHARE_DELETE	0x00000004	Other open operations can be performed on the file for delete access.

The CreateDisposition parameter can contain one of the following values:

Name	Value	Meaning
FILE_SUPERSEDE	0x00000000	*FILE_SUPERSEDE*- Indicates that if the file already exists then it should be superseded by the specified file. If it does not already exist then it should be created.
FILE_OPEN	0x00000001	*FILE_OPEN* - Indicates that if the file already exists it should be opened rather than creating a new file. If the file does not already exist then the operation should fail.
FILE_CREATE	0x00000002	*FILE_CREATE* - Indicates that if the file already exists then the operation should fail. If the file does not already exist then it should be created.
FILE_OPEN_IF	0x00000003	*FILE_OPEN_IF* - Indicates that if the file already exists, it should be opened. If the file does not already exist then it should be created.
FILE_OVERWRITE	0x00000004	*FILE_OVERWRITE* - Indicates that if the file already exists it should be opened and overwritten. If the file does not already exist then the operation should fail.
FILE_OVERWRITE_IF	0x00000005	*FILE_OVERWRITE_IF* - Indicates that if the file already exists it should be opened and overwritten. If the file does not already exist then it should be created.

Name	Value	Meaning
FILE_MAXIMUM_DISPOSITION	0x00000005	?

The ImpersonationLevel parameter can contain one or more of the following values:

Name	Value	Meaning
SECURITY_ANONYMOUS	0	Impersonation of the client at the Anonymous level
SECURITY_IDENTIFICATION	1	Impersonation of the client at the Identification level
SECURITY_IMPERSONATION	2	Impersonation of the client at the Impersonation level
SECURITY_DELEGATION	3	Impersonation of the client at the Delegation level

The SecurityFlags parameter can have either of the following two flags set:

Name	Value	Meaning
SECURITY_CONTEXT_TRACKING	0x00040000	Specifies that the security tracking mode is dynamic. If this flag is not specified, Security Tracking Mode is static.
SECURITY_EFFECTIVE_ONLY	0x00080000	Specifies that only the enabled aspects of the client's security context are available to the server. If this flag is not specified, all aspects of the client's security context are available. This flag allows the client to limit the groups and privileges that a server can use while impersonating the client.

The server response to the NT_CREATE_ANDX request is as follows:

```
Server Response                 Description
===============                 ============
UCHAR WordCount;                Count of parameter words = 26
UCHAR AndXCommand;              0xFF = None
UCHAR AndXReserved;             MUST BE ZERO
USHORT AndXOffset;              Offset to next command WordCount
UCHAR OplockLevel;              The oplock level granted:
                                        0 - No oplock granted
                                        1 - Exclusive oplock granted
                                        2 - Batch oplock granted
                                        3 - Level II oplock granted
USHORT Fid;                     The file ID
ULONG CreateAction;             The action taken
TIME CreationTime;              The time the file was created
TIME LastAccessTime;            The time the file was accessed
TIME LastWriteTime;             The time the file was last written
TIME ChangeTime;                The time the file was last changed
ULONG ExtFileAttributes;        The file attributes
LARGE_INTEGER AllocationSize;   The number of byes allocated
LARGE_INTEGER EndOfFile;        The end of file offset
USHORT FileType;
USHORT DeviceState;             State of IPC device (e.g. pipe)
BOOLEAN Directory;              TRUE if this is a directory
USHORT ByteCount;               = 0
```

```
The following SMBs may follow SMB_COM_NT_CREATE_ANDX:
SMB_COM_READ    SMB_COM_READ_ANDX
SMB_COM_IOCTL
```

4.2.1.1. Errors

```
ERRDOS codes
------------
ERRbadfile
ERRbadpath
ERRnofids
ERRnoaccess
ERRnomem
ERRbadaccess
ERRbadshare
ERRfileexists
ERRquota

ERRSRV codes
------------
ERRaccess
ERRinvdevice
ERRinvtid
ERRbaduid
```

4.2.2. NT_TRANSACT_CREATE: Create or Open File with EAs or SD

This command is used to create or open a file or a directory, when EAs or an SD must be applied to the file. The parameter and data blocks for the client's CREATE request include the following data:

Request Parameter Block Encoding	Description
ULONG Flags;	Creation flags (see below)
ULONG RootDirectoryFid;	Optional directory for relative open
ACCESS_MASK DesiredAccess;	Access desired (See Section 3.8 for an explanation of this field)
LARGE_INTEGER AllocationSize;	The initial allocation size in bytes, if file created
ULONG ExtFileAttributes;	The extended file attributes
ULONG ShareAccess;	The share access
ULONG CreateDisposition;	Action if file does/does not exist
ULONG CreateOptions;	Options for creating a new file
ULONG SecurityDescriptorLength;	Length of SD in bytes
ULONG EaLength;	Length of EA in bytes
ULONG NameLength;	Length of name in characters
ULONG ImpersonationLevel;	Security QOS information
UCHAR SecurityFlags;	Security QOS information
STRING Name[NameLength];	The name of the file (not NULL terminated)

```
Request Data Block Encoding          Description
 ============================         ============
  UCHAR SecurityDescriptor[
     SecurityDescriptorLength];
  UCHAR ExtendedAttributes[EaLength];
```

The Flags parameter can contain one of the following values:

Creation Flags Name	Value	Description
NT_CREATE_REQUEST_OPLOCK	0x02	Exclusive oplock requested
NT_CREATE_REQUEST_OPBATCH	0x04	Batch oplock requested
NT_CREATE_OPEN_TARGET_DIR	0x08	Target for open is a directory

The parameter block of the server response is defined as:

```
Response Parameter Block Encoding    Description
==================================   ============
 UCHAR OplockLevel;                   The oplock level granted
 UCHAR Reserved;
 USHORT Fid;                          The file ID
 ULONG CreateAction;                  The action taken
 ULONG EaErrorOffset;                 Offset of the EA error
 TIME CreationTime;                   The time the file was created
 TIME LastAccessTime;                 The time the file was accessed
 TIME LastWriteTime;                  The time the file was last written
 TIME ChangeTime;                     The time the file was last changed
 ULONG ExtFileAttributes;             The file attributes
 LARGE_INTEGER AllocationSize;        The number of byes allocated
 LARGE_INTEGER EndOfFile;             The end of file offset
 USHORT FileType;
 USHORT DeviceState;                  State of IPC device (e.g. pipe)
 BOOLEAN Directory;                   TRUE if this is a directory
```

See the description of NT_CREATE_ANDX (section 4.2.1) for further definition of the CREATE request/response parameters.

4.2.2.1. Errors

```
ERRDOS codes
------------
ERRbadfile
ERRbadpath
ERRnofids
ERRnoaccess
ERRnomem
ERRbadaccess
ERRbadshare
ERRfileexists
ERRquota

ERRSRV codes
------------
ERRaccess
```

```
ERRinvdevice
ERRinvtid
ERRbaduid
```

4.2.3. CREATE_TEMPORARY: Create Temporary File

The server creates a data file in the specified Directory, relative to Tid in the SMB header, and assigns a unique name to it. The client request and server response for the command are:

```
Client Request                  Description
==============                  ============
 UCHAR WordCount;                Count of parameter words = 3
 USHORT reserved;                Ignored by the server
 UTIME CreationTime;             New file's creation time stamp
 USHORT ByteCount;               Count of data bytes; min = 2
 UCHAR BufferFormat;             0x04
 STRING DirectoryName[];         Directory name

Server Response                 Description
===============                 ============
 UCHAR WordCount;                Count of parameter words = 1
 USHORT Fid;                     File handle
 USHORT ByteCount;               Count of data bytes; min = 2
 UCHAR BufferFormat;             0x04
 STRING Filename[];              File name
```

Fid is the returned handle for future file access. Filename is the name of the file that was created within the requested Directory. It is opened in compatibility mode with read/write access for the client.

Support of CreationTime by the server is optional.

4.2.3.1. Errors

```
ERRDOS codes
------------
ERRbadfile
ERRbadpath
ERRnofids
ERRnoaccess
ERRnomem
ERRbadaccess
ERRbadshare
ERRfileexists
ERRquota

ERRSRV codes
------------
ERRaccess
ERRinvdevice
ERRinvtid
ERRbaduid
```

4.2.4. READ_ANDX: Read Bytes

Client requests a file read, using the SMB fields specified below:

```
Client Request              Description
==============             ============
 UCHAR WordCount;           Count of parameter words = 10 or 12
 UCHAR AndXCommand;         Secondary (X) command; 0xFF = none
 UCHAR AndXReserved;        Reserved (must be 0)
 USHORT AndXOffset;         Offset to next command WordCount
 USHORT Fid;                File handle
 ULONG Offset;              Offset in file to begin read
 USHORT MaxCount;           Max number of bytes to return
 USHORT MinCount;           Reserved for obsolescent requests
 ULONG MaxCountHigh;        High 16 bits of MaxCount if
                             CAP_LARGE_READX; else MUST BE ZERO
 USHORT Remaining;          Reserved for obsolescent requests
 ULONG OffsetHigh;          Upper 32 bits of offset (only if
                             WordCount is 12)
 USHORT ByteCount;          Count of data bytes = 0
```

And, the server response is:

```
Server Response             Description
===============             ============
 UCHAR WordCount;            Count of parameter words = 12
 UCHAR AndXCommand;          Secondary (X) command; 0xFF = none
 UCHAR AndXReserved;         Reserved (must be 0)
 USHORT AndXOffset;          Offset to next command WordCount
 USHORT Remaining;           Reserved -- must be -1
 USHORT DataCompactionMode;
 USHORT Reserved;            Reserved (must be 0)
 USHORT DataLength;          Number of data bytes (min = 0)
 USHORT DataOffset;          Offset (from header start) to data
 USHORT DataLengthHigh;      High 16 bits of number of data bytes if
                              CAP_LARGE_READX; else MUST BE ZERO
 USHORT Reserved[4];         Reserved (must be 0)
 USHORT ByteCount;           Count of data bytes; ignored if
                              CAP_LARGE_READX
 UCHAR Pad[];
 UCHAR Data[DataLength];     Data from resource
```

If the file specified by Fid has any portion of the range specified by Offset and MaxCount locked for exclusive use by a client with a different connection or Pid, the request will fail with ERRlock.

If the negotiated dialect is NT LM 0.12 or later, the client may use the 12 parameter word version of the request. This version allows specification of 64 bit file offsets.

If CAP_LARGE_READX was indicated by the server in the negotiate protocol response, the request's MaxCount field may exceed the negotiated buffer size if Fid refers to a disk file. The server may arbitrarily elect to return fewer than MaxCount bytes in response.

The SMB server MAY use the MinCount on named-pipe calls to determine if this is a blocking read or a non-blocking read. (Non blocking is determined by MinCount = 0). Note that for blocking reads, the length

required to succeed is actually the ReadLength and not the MinCount. (So in some sense, MinCount has become more of an indicator of blocking vs. non-blocking rather than a true length)

The following SMBs may follow SMB_COM_READ_ANDX:

```
SMB_COM_CLOSE
```

4.2.4.1. Errors

```
ERRDOS/ERRnoaccess
ERRDOS/ERRbadfid
ERRDOS/ERRlock
ERRDOS/ERRbadaccess
ERRSRV/ERRinvid
ERRSRV/ERRbaduid
```

4.2.5. WRITE_ANDX: Write Bytes to file or resource

Client requests a file write, using the SMB fields specified below:

```
Client Request            Description
==============            ============
UCHAR WordCount;          Count of parameter words = 12 or 14
UCHAR AndXCommand;        Secondary (X) command; 0xFF = none
UCHAR AndXReserved;       Reserved (must be 0)
USHORT AndXOffset;        Offset to next command WordCount
USHORT Fid;               File handle
ULONG Offset;             Offset in file to begin write
ULONG Reserved;           Must be 0
USHORT WriteMode;         Write mode bits:
                               0 - write through
USHORT Remaining;         Bytes remaining to satisfy request
USHORT DataLengthHigh;    High 16 bits of data length if
                               CAP_LARGE_WRITEX; else MUST BE ZERO
USHORT DataLength;        Number of data bytes in buffer (>=0)
USHORT DataOffset;        Offset to data bytes
ULONG OffsetHigh;         Upper 32 bits of offset (only present if
                               WordCount = 14)
USHORT ByteCount;         Count of data bytes; ignored if
                               CAP_LARGE_WRITEX
UCHAR Pad[];              Pad to SHORT or LONG
UCHAR Data[DataLength];   Data to write
```

And, the server response is:

```
Server Response           Description
================          ============
UCHAR WordCount;          Count of parameter words = 6
UCHAR AndXCommand;        Secondary (X) command; 0xFF = none
UCHAR AndXReserved;       Reserved (must be 0)
USHORT AndXOffset;        Offset to next command WordCount
USHORT Count;             Number of bytes written
USHORT Remaining;         Reserved
ULONG Reserved;
USHORT ByteCount;         Count of data bytes = 0
```

If the file specified by Fid has any portion of the range specified by Offset and MaxCount locked for shared or exclusive use by a client with a different connection or Pid, the request will fail with ERRlock.

A ByteCount of 0 does not truncate the file. Rather a zero length write merely transfers zero bytes of information to the file. A request such as SMB_COM_WRITE must be used to truncate the file.

If WriteMode has bit0 set in the request and Fid refers to a disk file, the response is not sent from the server until the data is on stable storage.

If the negotiated dialect is NT LM 0.12 or later, the 14 word format of this SMB may be used to access portions of files requiring offsets expressed as 64 bits. Otherwise, the OffsetHigh field must be omitted from the request.

If CAP_LARGE_WRITEX was indicated by the server in the negotiate protocol response, the request's DataLength field may exceed the negotiated buffer size if Fid refers to a disk file.

The following are the valid AndXCommand values for this SMB:

```
SMB_COM_READ              SMB_COM_READ_ANDX
SMB_COM_LOCK_AND_READ     SMB_COM_WRITE_ANDX
SMB_COM_CLOSE
```

4.2.5.1. Errors

```
ERRDOS/ERRnoaccess
ERRDOS/ERRbadfid
ERRDOS/ERRlock
ERRDOS/ERRbadaccess
ERRSRV/ERRinvid
ERRSRV/ERRbaduid
```

4.2.6. LOCKING_ANDX: Lock or Unlock Byte Ranges

SMB_COM_LOCKING_ANDX allows both locking and/or unlocking of file range(s). A description of the fields of the client request, and explanations for several of the fields are provided below.

```
Client Request                    Description
===============                   ============
 UCHAR WordCount;                  Count of parameter words = 8
 UCHAR AndXCommand;                Secondary (X) command; 0xFF = none
 UCHAR AndXReserved;               Reserved (must be 0)
 USHORT AndXOffset;                Offset to next command WordCount
 USHORT Fid;                       File handle
 UCHAR LockType;                   See LockType table below
 UCHAR OplockLevel;                The new oplock level
 ULONG Timeout;                    Milliseconds to wait for unlock
 USHORT NumberOfUnlocks;           Number of unlock range structures that
                                    follow
 USHORT NumberOfLocks;             Number of lock range structures that
                                           follow
 USHORT ByteCount;                 Count of data bytes
 LOCKING_ANDX_RANGE Unlocks[];     Unlock ranges
 LOCKING_ANDX_RANGE Locks[];       Lock ranges
```

The LockType parameter can take on one of the values in the following table:

Flag Name	Value	Description
LOCKING_ANDX_SHARED_LOCK	0x01	Read-only lock
LOCKING_ANDX_OPLOCK_RELEASE	0x02	Oplock break notification
LOCKING_ANDX_CHANGE_LOCKTYPE	0x04	Change lock type
LOCKING_ANDX_CANCEL_LOCK	0x08	Cancel outstanding request
LOCKING_ANDX_LARGE_FILES	0x10	Large file locking format

The format for LOCKING_ANDX_RANGE is:

```
USHORT Pid;                     PID of process "owning" lock
ULONG Offset;                   Offset to bytes to [un]lock
ULONG Length;                   Number of bytes to [un]lock
```

And, for a large file, it is:

```
USHORT Pid;                     PID of process "owning" lock
USHORT Pad;                     Pad to DWORD align (Must be zero)
ULONG OffsetHigh;               Offset to bytes to [un]lock (high)
ULONG OffsetLow;                Offset to bytes to [un]lock (low)
ULONG LengthHigh;               Number of bytes to [un]lock
                                  (high)
ULONG LengthLow;                Number of bytes to [un]lock (low)
```

The server response is:

```
Server Response                 Description
===============                 ===========
 UCHAR WordCount;                Count of parameter words = 2
 UCHAR AndXCommand;              Secondary (X) command; 0xFF = none
 UCHAR AndXReserved;             Reserved (must be 0)
 USHORT AndXOffset;              Offset to next command WordCount
 USHORT ByteCount;               Count of data bytes = 0
```

Locking is a simple mechanism for excluding other processes read/write access to regions of a file. The locked regions can be anywhere in the logical file. Locking beyond end-of-file is permitted. Lock conflicts (overlapping lock-requests) should cause the server to refuse the lock to the latter requestor. Any process using the *Fid* specified in this request's *Fid* has access to the locked bytes; other processes will be denied the locking of the same bytes.

The proper method for using locks is not to rely on being denied read or write access on any of the read/write protocols but rather to attempt the locking protocol and proceed with the read/write only if the locks succeeded.

Locking a range of bytes will fail if any subranges or overlapping ranges are locked, if the PID/UID of the requestor is not the same, and the locks are not compatible. In other words, if any of the specified bytes are already locked, the lock will fail.

If NumberOfUnlocks is non-zero, the Unlocks vector contains NumberOfUnlocks elements. Each element requests that a lock at Offset of Length be released. If NumberOfLocks is nonzero, the

Locks vector contains NumberOfLocks elements. Each element requests the acquisition of a lock at Offset of Length.

Timeout is the maximum amount of time to wait for the byte range(s) specified to become unlocked. A timeout value of 0 indicates that the server should fail immediately if any lock range specified is locked. A timeout value of -1 indicates that the server should wait as long as it takes for each byte range specified to become unlocked so that it may be again locked by this protocol. Any other value of smb_timeout specifies the maximum number of milliseconds to wait for all lock range(s) specified to become available.

If any of the lock ranges timeout because of the area to be locked is already locked (or the lock fails), the other ranges in the protocol request which were successfully locked as a result of this protocol will be unlocked (either all requested ranges will be locked when this protocol returns to the client or none).

If LockType has the LOCKING_ANDX_SHARED_LOCK flag set, the lock is specified as a shared lock. Locks for both read and write (where LOCKING_ANDX_SHARED_LOCK is clear) should be prohibited, but other shared locks should be permitted. If shared locks can not be supported by a server, the server should map the lock to a lock for both read and write. Closing a file with locks still in force causes the locks to be released in no defined order.

If LockType has the LOCKING_ANDX_LARGE_FILES flag set and if the negotiated protocol is NT LM 0.12 or later, then the Locks and Unlocks vectors are in the Large File LOCKING_ANDX_RANGE format. This allows specification of 64 bit offsets for very large files.

If the one and only member of the Locks vector has the LOCKING_ANDX_CANCEL_LOCK flag set in the LockType field, the client is requesting the server to cancel a previously requested, but not yet responded to, lock.

If LockType has the LOCKING_ANDX_CHANGE_LOCKTYPE flag set, the client is requesting that the server atomically change the lock type from a shared lock to an exclusive lock or vice versa. If the server can not do this in an atomic fashion, the server must reject this request. (Note: Windows NT and Windows 95 servers do not support this capability.)

If the client sends an SMB_LOCKING_ANDX SMB with the LOCKING_ANDX_OPLOCK_RELEASE flag set and *NumberOfLocks* is zero, the server does not send a response. The entire message sent and received including the optional second protocol must fit in the negotiated maximum transfer size. The following are the only valid SMB commands for AndXCommand for SMB_COM_LOCKING_ANDX:

```
SMB_COM_READ        SMB_COM_READ_ANDX
SMB_COM_WRITE       SMB_COM_WRITE_ANDX
SMB_COM_FLUSH
```

4.2.6.1. Errors

```
ERRDOS/ERRbadfile
ERRDOS/ERRbadfid
ERRDOS/ERRlock
ERRDOS/ERRinvdevice
ERRSRV/ERRinvid
ERRSRV/ERRbaduid
```

4.2.7. SEEK: Seek in File

The seek message is sent to set the current file pointer for *Fid*.

```
Client Request                  Description
==============                  ================================
UCHAR WordCount;                Count of parameter words = 4
 USHORT Fid;                     File handle
 USHORT Mode;                      Seek mode:
                                         0 = from start of file
                                         1 = from current position
                                         2 = from end of file
 LONG Offset;                      Relative offset
 USHORT ByteCount;               Count of data bytes = 0
```

The "current position" reflects the offset plus data length specified in the previous read, write, or seek request; and the pointer set by this command will be replaced by the offset specified in the next read, write, or seek command.

```
Server Response                 Description
================                ============
 UCHAR WordCount;                Count of parameter words = 2
 ULONG Offset;                          Offset from start of file
 USHORT ByteCount;               Count of data bytes = 0
```

The response returns the new file pointer in *Offset,* which is expressed as the offset from the start of the file, and may be beyond the current end of file. An attempt to seek to before the start of file sets the current file pointer to start of the file.

This request should generally be issued only by clients wishing to find the size of a file, because all read and write requests include the read or write file position as part of the SMB. This request is inappropriate for very large files, as the offsets specified are only 32 bits. A seek that results in an Offset that cannot be expressed in 32 bits returns the least significant.

4.2.7.1. Errors

```
ERRDOS/ERRbadfid
ERRDOS/ERRnoaccess
ERRSRV/ERRinvdevice
ERRSRV/ERRinvid
ERRSRV/ERRbaduid
```

4.2.8. FLUSH: Flush File

The flush SMB is sent to ensure all data and allocation information for the corresponding file has been written to stable storage. When the Fid has a value -1 (hex FFFF), the server performs a flush for all file handles associated with the client and Pid. The response is not sent until the writes are complete.

```
Client Request                  Description
==============                  ================================
 UCHAR WordCount;                Count of parameter words = 1
 USHORT Fid;                     File handle
 USHORT ByteCount;               Count of data bytes = 0
```

This client request is probably expensive to perform at the server, since the server's operating system is generally scheduling disk writes is a way which is optimal for the system's read and write activity integrated over the entire population of clients. This message from a client "interferes" with the server's ability to optimally schedule the disk activity; clients are discouraged from overuse of this SMB request.

```
Server Response                 Description
================                ============
 UCHAR WordCount;                Count of parameter words = 0
 USHORT ByteCount;               Count of data bytes = 0
```

4.2.8.1. Errors

```
ERRDOS/ERRbadfid
ERRSRV/ERRinvid
ERRSRV/ERRbaduid
```

4.2.9. CLOSE: Close File

The close message is sent to invalidate a file handle for the requesting process. All locks or other resources held by the requesting process on the file should be released by the server. The requesting process can no longer use Fid for further file access requests.

```
Client Request                  Description
==============                  ============
 UCHAR WordCount;                Count of parameter words = 3
 USHORT Fid;                     File handle
 UTIME LastWriteTime;            Time of last write
 USHORT ByteCount;               Count of data bytes = 0
```

If LastWriteTime is 0, the server should allow its local operating system to set the file's times. Otherwise, the server should set the time to the values requested. Failure to set the times, even if requested by the client in the request message, should not result in an error response from the server.

If Fid refers to a print spool file, the file should be spooled to the printer at this time.

```
Server Response                 Description
================                ============
 UCHAR WordCount;                Count of parameter words = 0
 USHORT ByteCount;               Count of data bytes = 0
```

4.2.9.1. Errors

```
ERRDOS/ERRbadfid
ERRSRV/ERRinvdevice
ERRSRV/ERRinvid
ERRSRV/ERRbaduid
```

4.2.10. CLOSE_AND_TREE_DISCONNECT

Close the file and perform a tree disconnect.

The close and tree disconnect message is sent to close a file and perform a tree disconnect. All locks or other resources held by the requesting process on the file should be released by the server. The requesting process can no longer use Fid for further file access requests. The server

will perform a TREE_DISCONNECT after completing the close operation. The requesting process can no longer use Tid for further access requests.

```
Client Request                  Description
==============                  ============
 UCHAR WordCount;                Count of parameter words = 3
 USHORT Fid;                     File handle
 UTIME LastWriteTime;            Time of last write
 USHORT ByteCount;               Count of data bytes = 0
```

If LastWriteTime is 0, the server should allow its local operating system to set the file's times. Otherwise, the server should set the time to the values requested. Failure to set the times, even if requested by the client in the request message, should not result in an error response from the server.

If Fid refers to a print spool file, the file should be spooled to the printer at this time.

```
Server Response                 Description
================                ============
 UCHAR WordCount;                Count of parameter words = 0
 USHORT ByteCount;               Count of data bytes = 0
```

4.2.10.1. Errors

```
ERRDOS/ERRbadfid
ERRSRV/ERRinvdevice
ERRSRV/ERRinvid
ERRSRV/ERRbaduid
```

4.2.11. DELETE: Delete File

The delete file message is sent to delete a data file. The appropriate Tid and additional pathname are passed. Read only files may not be deleted, the read-only attribute must be reset prior to file deletion.

```
Client Request                  Description
==============                  ============
 UCHAR WordCount;                Count of parameter words = 1
 USHORT SearchAttributes;
 USHORT ByteCount;               Count of data bytes; min = 2
 UCHAR BufferFormat;             0x04
 STRING FileName[];              File name
```

Multiple files may be deleted in response to a single request as SMB_COM_DELETE supports wildcards.

SearchAttributes indicates the attributes that the target file(s) must have. If the attribute is zero then only normal files are deleted. If the system file or hidden attributes are specified, then the delete is inclusive - both the specified type(s) of files and normal files are deleted. File attributes are described in the "Attribute Encoding" section (3.11) of this document.

If bit0 of the Flags2 field of the SMB header is set, a pattern is passed in, and the file has a long name, then the passed pattern must match the long file name for the delete to succeed. If bit0 is clear, a pattern is passed in, and the file has a long name, then the passed pattern must match the file's short name for the deletion to succeed.

```
Server Response                 Description
================                ============
 UCHAR WordCount;                Count of parameter words = 0
 USHORT ByteCount;               Count of data bytes = 0
```

4.2.11.1. Errors

```
ERRDOS/ERRbadpath
ERRDOS/ERRbadfile
ERRDOS/ERRnoaccess
ERRHRD/ERRnowrite
ERRSRV/ERRaccess
ERRSRV/ERRinvdevice
ERRSRV/ERRinvid
ERRSRV/ERRbaduid
```

4.2.12. RENAME: Rename File

The rename file message is sent to change the name of a file.

```
Client Request                  Description
================                ============
 UCHAR WordCount;                Count of parameter words = 1
 USHORT SearchAttributes;        Target file attributes
 USHORT ByteCount;               Count of data bytes; min = 4
 UCHAR BufferFormat1;            0x04
 STRING OldFileName[];           Old file name
 UCHAR BufferFormat2;            0x04
 STRING NewFileName[];           New file name
```

The file, OldFileName, must exist and NewFileName must not. Both pathnames must be relative to the Tid specified in the request. Open files may be renamed.

Multiple files may be renamed in response to a single request as Rename File supports wildcards in the file name (last component of the pathname).

SearchAttributes indicates the attributes that the target file(s) must have. If SearchAttributes is zero then only normal files are renamed. If the system file or hidden attributes are specified then the rename is inclusive - both the specified type(s) of files and normal files are renamed. The encoding of SearchAttributes is described in section 3.11 - File Attribute Encoding.

```
Server Response                 Description
================                ============
 UCHAR WordCount;                Count of parameter words = 0
 USHORT ByteCount;               Count of data bytes = 0
```

4.2.12.1. Errors

```
ERRDOS/ERRbadpath
ERRDOS/ERRbadfile
ERRDOS/ERRnoaccess
ERRDOS/ERRdiffdevice
ERRHRD/ERRnowrite
ERRSRV/ERRaccess
ERRSRV/ERRinvdevice
```

```
ERRSRV/ERRinvid
ERRSRV/ERRbaduid
```

4.2.13. NT_RENAME:

The rename file message is sent to change the name of a file. This version of RENAME supports NT link tracking info.

```
Client Request                  Description
==============                  ============
 UCHAR WordCount;                Count of parameter words = 4
 USHORT SearchAttributes;
 USHORT Information Level;
 ULONG ClusterCount;
 USHORT ByteCount;               Count of data bytes; min = 4
 UCHAR Buffer[1];                 Buffer containing:
                                   UCHAR BufferFormat1 0x04 -- ASCII
                                   UCHAR OldFileName[] Old file name
                                   UCHAR BufferFormat2 0x04 -- ASCII
                                   UCHAR NewFileName[] New file name

Server Response                 Description
================                ============
 UCHAR WordCount;                Count of parameter words = 0
 USHORT ByteCount;               Count of data bytes = 0
 UCHAR Buffer[1];                 empty
```

Non-NT machines can ignore the extra parameters (InfoLevel, SearchAttributes, ClusterCount) and just perform a normal rename.

4.2.13.1. Errors

```
ERRDOS codes
------------
ERRbadfile
ERRbadpath
ERRnofids
ERRnoaccess
ERRnomem
ERRfileexists

ERRSRV codes
------------
ERRaccess
ERRinvdevice
ERRinvtid
ERRbaduid
```

4.2.14. MOVE: Rename File

The source file is copied to the destination and the source is subsequently deleted.

```
Client Request                  Description
==============                  ============
 UCHAR WordCount;                Count of parameter words = 3
```

```
USHORT Tid2;                    Second (target) file id
USHORT OpenFunction;            What to do if target file exists
USHORT Flags;                   Flags to control move operations:
                                 0 - target must be a file
                                 1 - target must be a directory
                                 2 - reserved (must be 0)
                                 3 - reserved (must be 0)
                                 4 - verify all writes
USHORT ByteCount;               Count of data bytes; min = 2
UCHAR Format1;                  0x04
STRING OldFileName[];           Old file name
UCHAR FormatNew;                0x04
STRING NewFileName[];           New file name
```

OldFileName is copied to NewFileName, then OldFileName is deleted. Both OldFileName and NewFileName must refer to paths on the same server. NewFileName can refer to either a file or a directory. All file components except the last must exist; directories will not be created.

NewFileName can be required to be a file or a directory by the Flags field.

The Tid in the header is associated with the source while Tid2 is associated with the destination. These fields may contain the same or differing valid values. Tid2 can be set to -1 indicating that this is to be the same Tid as in the SMB header. This allows use of the move protocol with SMB_TREE_CONNECT_ANDX.

```
Server Response                 Description
===============                 ============
 UCHAR WordCount;                Count of parameter words = 1
 USHORT Count;                   Number of files moved
 USHORT ByteCount;               Count of data bytes; min = 0
 UCHAR ErrorFileFormat;          0x04  (only if error)
 STRING ErrorFileName[];         Pathname of file where error
                                   Occurred
```

The source path must refer to an existing file or files. Wildcards are permitted. Source files specified by wildcards are processed until an error is encountered. If an error is encountered, the expanded name of the file is returned in ErrorFileName. Wildcards are not permitted in NewFileName.

OpenFunction controls what should happen if the destination file exists. If (OpenFunction & 0x30) == 0, the operation should fail if the destination exists. If (OpenFunction & 0x30) == 0x20, the destination file should be overwritten.

4.2.14.1. Errors

```
ERRDOS/ERRfilexists
ERRDOS/ERRbadfile
ERRDOS/ERRnoaccess
ERRDOS/ERRnofiles
ERRDOS/ERRbadshare
ERRHRD/ERRnowrite
ERRSRV/ERRnoaccess
ERRSRV/ERRinvdevice
ERRSRV/ERRinvid
```

```
ERRSRV/ERRbaduid
ERRSRV/ERRnosupport
ERRSRV/ERRaccess
```

4.2.15. COPY: Copy File

The source file is copied to the target.

```
Client Request                  Description
==============                  ============
 UCHAR WordCount;                Count of parameter words = 3
 USHORT Tid2;                    Second (target) path TID
 USHORT OpenFunction;            What to do if target file exists
 USHORT Flags;                   Flags to control copy operation:
                                  bit 0 - target must be a file
                                  bit 1 - target must be a dir.
                                  bit 2 - copy target mode:
                                   0 = binary, 1 = ASCII
                                  bit 3 - copy source mode:
                                   0 = binary, 1 = ASCII
                                  bit 4 - verify all writes
                                  bit 5 - tree copy
 USHORT ByteCount;               Count of data bytes; min = 2
 UCHAR SourceFileNameFormat;     0x04
 STRING SourceFileName;          Pathname of source file
 UCHAR TargetFileNameFormat;     0x04
 STRING TargetFileName;          Pathname of target file
```

The file at SourceName is copied to TargetFileName, both of which must refer to paths on the same server.

The Tid in the header is associated with the source while Tid2 is associated with the destination. These fields may contain the same or differing valid values. Tid2 can be set to -1 indicating that this is to be the same Tid as in the SMB header. This allows use of the move protocol with SMB_TREE_CONNECT_ANDX.

```
Server Response                 Description
===============                 ============
 UCHAR WordCount;                Count of parameter words = 1
 USHORT Count;                   Number of files copied
 USHORT ByteCount;               Count of data bytes; min = 0
 UCHAR ErrorFileFormat;          0x04 (only if error)
 STRING ErrorFileName;
```

The source path must refer to an existing file or files. Wildcards are permitted. Source files specified by wildcards are processed until an error is encountered. If an error is encountered, the expanded name of the file is returned in ErrorFileName. Wildcards are not permitted in TargetFileName. TargetFileName can refer to either a file or a directory.

The destination can be required to be a file or a directory by the bits in Flags. If neither bit0 nor bit1 are set, the destination may be either a file or a directory. The Flags field also controls the copy mode. In a binary copy for the source, the copy stops the first time an EOF (control-Z) is encountered. In a binary copy for the target, the server must make sure that there is exactly one EOF in the target file and that it is the last character of the file.

If the destination is a file and the source contains wildcards, the destination file will either be truncated or appended to at the start of the operation depending on bits in OpenFunction (see section 3.7). Subsequent files will then be appended to the file.

If the negotiated dialect is LM1.2X002 or later, bit5 of Flags is used to specify a tree copy on the remote server. When this option is selected the destination must not be an existing file and the source mode must be binary. A request with bit5 set and either bit0 or bit3 set is therefore an error. When the tree copy mode is selected, the Count field in the server response is undefined.

4.2.15.1. Errors

```
ERRDOS/ERRfilexists
ERRDOS/ERRshare
ERRDOS/ERRnofids
ERRDOS/ERRbadfile
ERRDOS/ERRnoaccess
ERRDOS/ERRnofiles
ERRDOS/ERRbadshare
ERRSRV/ERRnoaccess
ERRSRV/ERRinvdevice
ERRSRV/ERRinvid
ERRSRV/ERRbaduid
ERRSRV/ERRaccess
```

4.2.16. TRANS2_QUERY_PATH_INFORMATION: Get File Attributes Given Path

This request is used to get information about a specific file or subdirectory.

```
Client Request             Value
===============            ======
 WordCount                  15
 MaxSetupCount              0
 SetupCount                 1
 Setup[0]                   TRANS2_QUERY_PATH_INFORMATION
```

The request's parameter block uses the following format:

```
 Parameter Block Encoding   Description
 ========================  ============
  USHORT InformationLevel;   Level of information requested
  ULONG Reserved;            Must be zero
  STRING FileName;           File or directory name
```

InformationLevels are specified using these values:

InformationLevel	Value
SMB_INFO_STANDARD	1
SMB_INFO_QUERY_EA_SIZE	2
SMB_INFO_QUERY_EAS_FROM_LIST	3
SMB_INFO_QUERY_ALL_EAS	4
SMB_INFO_IS_NAME_VALID	6
SMB_QUERY_FILE_BASIC_INFO	0x101

InformationLevel	Value
SMB_QUERY_FILE_STANDARD_INFO	0x102
SMB_QUERY_FILE_EA_INFO	0x103
SMB_QUERY_FILE_NAME_INFO	0x104
SMB_QUERY_FILE_ALL_INFO	0x107
SMB_QUERY_FILE_ALT_NAME_INFO	0x108
SMB_QUERY_FILE_STREAM_INFO	0x109
SMB_QUERY_FILE_COMPRESSION_INFO	0x10B
SMB_QUERY_FILE_UNIX_BASIC	0x200
SMB_QUERY_FILE_UNIX_LINK	0x201

The requested information is placed in the Data portion of the transaction response. For the information levels greater than 0x100, the transaction response has 1 parameter word which should be ignored by the client.

The following sections describe the InformationLevel dependent encoding of the data part of the transaction response.

4.2.16.1. SMB_INFO_STANDARD & SMB_INFO_QUERY_EA_SIZE

```
Data Block Encoding              Description
====================             ============
 SMB_DATE CreationDate;           Date when file was created
 SMB_TIME CreationTime;           Time when file was created
 SMB_DATE LastAccessDate;         Date of last file access
 SMB_TIME LastAccessTime;         Time of last file access
 SMB_DATE LastWriteDate;          Date of last write to the file
 SMB_TIME LastWriteTime;          Time of last write to the file
 ULONG  DataSize;                 File Size
 ULONG AllocationSize;            Size of filesystem allocation unit
 USHORT Attributes;               File Attributes
 ULONG EaSize;                    Size of file's EA information
                                   (SMB_INFO_QUERY_EA_SIZE)
```

4.2.16.2. SMB_INFO_QUERY_EAS_FROM_LIST & SMB_INFO_QUERY_ALL_EAS

```
Response Field          Value
===============         ======
 MaxDataCount            Length of EAlist found (minimum value is 4)

Parameter Block
Encoding                Description
================        ============
 USHORT EaErrorOffset;  Offset into EAList of EA error

Data Block Encoding     Description
====================    ============
 ULONG ListLength;     Length of the remaining data
 UCHAR EaList[];       The extended attributes list
```

4.2.16.3. SMB_INFO_IS_NAME_VALID

This requests checks to see if the name of the file contained in the request's Data field has a valid path syntax. No parameters or data are returned on this information request. An error is returned if the syntax of the name is incorrect. Success indicates the server accepts the path syntax, but it does not ensure the file or directory actually exists.

4.2.16.4. SMB_QUERY_FILE_BASIC_INFO

```
Data Block Encoding              Description
====================             ============
 TIME CreationTime;             Time when file was created
 TIME  LastAccessTime;           Time of last file access
 TIME  LastWriteTime;            Time of last write to the file
 TIME  ChangeTime;               Time when file was last changed
 ULONG Attributes;                 File Attributes
ULONG Pad;                      Undefined
```

The valid file attributes are:

Attribute	Value	Description
FILE_ATTRIBUTE_READONLY	0x00000001	The file is read only. Applications can read the file but cannot write to it or delete it.
FILE_ATTRIBUTE_HIDDEN	0x00000002	The file is hidden. It is not to be included in an ordinary directory listing.
FILE_ATTRIBUTE_SYSTEM	0x00000004	The file is part of or is used exclusively by the operating system.
FILE_ATTRIBUTE_VOLUMEID	0x00000008	The corresponding object represents a label for a filesystem object (obsolete)
FILE_ATTRIBUTE_DIRECTORY	0x00000010	The file is a directory.
FILE_ATTRIBUTE_ARCHIVE	0x00000020	The file is an archive file. Applications use this attribute to mark files for backup or removal.
FILE_ATTRIBUTE_DEVICE	0x00000040	The file is mapped to a device e.g. a printer or serial device.
FILE_ATTRIBUTE_NORMAL	0x00000080	The file has no other attributes set. This attribute is valid only if used alone. All other attributes override this attribute.
FILE_ATTRIBUTE_TEMPORARY	0x00000100	The file is being used for temporary storage. Applications should write to the file only if absolutely necessary. Most of the file's data remains in memory without being flushed to the media because the file will soon be deleted.
FILE_ATTRIBUTE_SPARSE_FILE	0x00000200	The file is a sparse file.
FILE_ATTRIBUTE_REPARSE_POINT	0x00000400	The file has an associated reparse point.
FILE_ATTRIBUTE_COMPRESSED	0x00000800	The file or directory is compressed. For a file, this means that all of the data in the file is compressed. For a directory, this means that compression is the default for newly created files and subdirectories.

Attribute	Value	Description
FILE_ATTRIBUTE_OFFLINE	0x00001000	The data of the file is not immediately available. This attribute indicates that the file data has been physically moved to offline storage. This attribute is used by Remote Storage, the hierarchical storage management software in Windows 2000. Applications should not arbitrarily change this attribute.
FILE_ATTRIBUTE_NOT CONTENT INDEXED	0x00002000	The file will not be indexed by the content indexing service.
FILE_ATTRIBUTE_ENCRYPTED	0x00004000	The file or directory is encrypted. For a file, this means that all data streams in the file are encrypted. For a directory, this means that encryption is the default for newly created files and subdirectories.

4.2.16.5. SMB_QUERY_FILE_STANDARD_INFO

```
Data Block Encoding              Description
====================             ============
 LARGE_INTEGER AllocationSize;    Allocated size of the file in number
                                   of bytes
 LARGE_INTEGER EndOfFile;         Offset to the first free byte in the
                                   file
 ULONG NumberOfLinks;             Number of hard links to the file
 BOOLEAN DeletePending;           Indicates whether the file is marked
                                   for deletion
 BOOLEAN Directory;               Indicates whether the file is a
                                   Directory
```

4.2.16.6. SMB_QUERY_FILE_EA_INFO

```
Data Block Encoding              Description
====================             ============
 ULONG EASize;                    Size of the file's extended
                                   attributes in number of bytes
```

4.2.16.7. SMB_QUERY_FILE_NAME_INFO

```
Data Block Encoding              Description
====================             ============
 ULONG FileNameLength;            Length of the file name in number of
                                   bytes
 STRING FileName;                 Name of the file
```

NOTE: Do not include the path to the file.

4.2.16.8. SMB_QUERY_FILE_ALL_INFO

```
Data Block Encoding              Description
====================             ============
 TIME  CreationTime;             Time when file was created
 TIME  LastAccessTime;           Time of last file access
 TIME  LastWriteTime;            Time of last write to the file
 TIME  ChangeTime;                Time when file was last changed
```

```
USHORT Attributes;              File Attributes
LARGE_INTEGER AllocationSize;   Allocated size of the file in number
                                 of bytes
LARGE_INTEGER EndOfFile;        Offset to the first free byte in the
                                 file
ULONG NumberOfLinks;            Number of hard links to the file
BOOLEAN DeletePending;          Indicates whether the file is marked
                                 for deletion
BOOLEAN Directory;              Indicates whether the file is a
                                 directory
LARGE_INTEGER IndexNumber;      A file system unique identifier
ULONG EASize;                   Size of the file's extended
                                 attributes in number of bytes
ULONG AccessFlags;              Access that a caller has to the
                                 file; Possible values and meanings
                                 are specified below
LARGE_INTEGER IndexNumber1;     A file system unique identifier
LARGE_INTEGER                  Current byte offset within the file
                                  CurrentByteOffset;
ULONG Mode;                     Current Open mode of the file handle
                                 to the file; possible values and
                                 meanings are detailed below
ULONG AlignmentRequirement;     Buffer Alignment required by device;
                                 possible values detailed below
ULONG FileNameLength;           Length of the file name in number of
                                 bytes
STRING FileName;                Name of the file
```

The AccessFlags specifies the access permissions a caller has to the file. It can have any suitable combination of the following values:

AccessFlag Name	Value	Meaning
FILE_READ_DATA	0x00000001	Data can be read from the file
FILE_WRITE_DATA	0x00000002	Data can be written to the file
FILE_APPEND_DATA	0x00000004	Data can be appended to the file
FILE_READ_EA	0x00000008	Extended attributes associated with the file can be read
FILE_WRITE_EA	0x00000010	Extended attributes associated with the file can be written
FILE_EXECUTE	0x00000020	Data can be read into memory from the file using system paging I/O
FILE_READ_ATTRIBUTES	0x00000080	Attributes associated with the file can be read
FILE_WRITE_ATTRIBUTES	0x00000100	Attributes associated with the file can be written
DELETE	0x00010000	The file can be deleted
READ_CONTROL	0x00020000	The access control list and ownership associated with the file can be read
WRITE_DAC	0x00040000	The access control list and ownership associated with the file can be written
WRITE_OWNER	0x00080000	Ownership information associated with the file can be written

AccessFlag Name	Value	Meaning
SYNCHRONIZE	0x00100000	The file handle can waited on to synchronize with the completion of an input/output request

The Mode field specifies the mode in which the file is currently opened. The possible values may be a suitable and logical combination of the following:

Mode Name	Value	Meaning
FILE_WRITE_THROUGH	0x00000002	File is opened in a mode where data is written to the file before the driver completes a write request
FILE_SEQUENTIAL_ONLY	0x00000004	All access to the file is sequential
FILE_SYNCHRONOUS_IO_ALERT	0x00000010	All operations on the file are performed synchronously
FILE_SYNCHRONOUS_IO_NONALERT	0x00000020	All operations on the file are to be performed synchronously. Waits in the system to synchronize I/O queuing and completion are not subject to alerts.

The AlignmentRequirement field specifies buffer alignment required by the device and can have any one of the following values:

AlignmentRequirement Name	Value	Meaning
FILE_BYTE_ALIGNMENT	0x00000000	The buffer needs to be aligned on a byte boundary
FILE_WORD_ALIGNMENT	0x00000001	The buffer needs to be aligned on a word boundary
FILE_LONG_ALIGNMENT	0x00000003	The buffer needs to be aligned on a 4 byte boundary
FILE_QUAD_ALIGNMENT	0x00000007	The buffer needs to be aligned on an 8 byte boundary
FILE_OCTA_ALIGNMENT	0x0000000F	The buffer needs to be aligned on a 16 byte boundary
FILE_32_BYTE_ALIGNMENT	0x0000001F	The buffer needs to be aligned on a 32 byte boundary
FILE_64_BYTE_ALIGNMENT	0x0000003F	The buffer needs to be aligned on a 64 byte boundary
FILE_128_BYTE_ALIGNMENT	0x0000007F	The buffer needs to be aligned on a 128 byte boundary
FILE_256_BYTE_ALIGNMENT	0x000000FF	The buffer needs to be aligned on a 256 byte boundary
FILE_512_BYTE_ALIGNMENT	0x000001FF	The buffer needs to be aligned on a 512 byte boundary

Extended attributes are used primarily by OS/2 Network Clients since OS/2 1.2a, but are an optional feature (I.e., filesystems and network servers are not required to support it). Extended attributes provided alternate data streams that are most commonly used by OS/2 client programs for the following purposes:

1) Storing the compiled form of a batch file (the first time a REXX program is run it is compiled on the fly and stored in extended attributes, subsequent runs use the compiled form)

2) Storing desktop attributes for folders and desktop objects for the OS/2Workplace Shell.

Supporting extended attributes is not mandatory in order to support OS/2 clients or to support the vast majority of OS/2 programs. Note that Windows NT Workstations can generate extended attribute request when requested by older programs (such as OS/2) and Windows NT servers do support requests to get or set extended attributes. Windows NT programs with needs to store "extended" attribute information, now largely use the capability to associate data streams with files that was introduced in NT 4. In both cases, the general concept is similar to the data fork concept

introduced by the Macintosh filesystem. Extended Attributes have been used for Macintosh compatibility in the past (to emulate data forks).

4.2.16.9. SMB_QUERY_FILE_ALT_NAME_INFO

Retrieves the 8.3 form of the file name, given the long name specified in the data block encoding.

```
Data Block Encoding              Description
====================             ============
 ULONG FileNameLength;           Length of the file name in number
                                  of bytes
 STRING FileName;                Name of the file
```

4.2.16.10. SMB_QUERY_FILE_STREAM_INFO

```
Data Block Encoding              Description
====================             ============
 ULONG NextEntryOffset;          Offset to the next entry (in bytes)
 ULONG StreamNameLength;         Length of the stream name in number
                                  of bytes
 LARGE_INTEGER StreamSize;       Size of the stream in number of
                                  bytes
 LARGE_INTEGER                   Allocated size of the stream in
  StreamAllocationSize;           number of bytes
 STRING FileName;                Name of the stream
```

NOTE: When more than one data block is returned, the NextEntryOffset is the offset to the next entry and is 0 for the last entry. STATUS_INVALID_PARAMETER is returned if file streams are not supported.

4.2.16.11. SMB_QUERY_FILE_COMPRESSION_INFO

```
Data Block Encoding              Description
====================             ============
 LARGE_INTEGER                    Size of the compressed file in
  CompressedFileSize;              number of bytes
 USHORT CompressionFormat;        A constant signifying the
                                   compression algorithm used. Possible
                                   values are:
                                   0 - There is no compression
                                   2- Compression Format is LZNT
 UCHAR CompressionUnitShift;
 UCHAR ChunkShift;                Stored in log2 format (1 << ChunkShift =
                                   ChunkSizeInBytes)
 UCHAR ClusterShift;              Indicates how much space must be
                                   saved to successfully compress a
                                   compression unit
 UCHAR Reserved[3];
```

4.2.16.12.SMB_QUERY_FILE_UNIX_BASIC

Used to retrieve UNIX specific file information

```
Data Block Encoding           Description
 ====================         ============
  LARGE_INTEGER EndOfFile;    File size
  LARGE_INTEGER NumOfBytes    Number of file system bytes used to store file
  TIME LastStatusChange;      Last time the status of the file was changed.
                                This is in DCE time.
  TIME LastAccessTime;        Time of last file access.  This is DCE time.
  TIME LastModificationTime;  Last modification time.  This is DCE time.
  LARGE_INTEGER Uid;          Numeric user id for the owner
  LARGE_INTEGER Gid;          Numeric group id of owner
  ULONG Type;                 Enumeration specifying the file type.
                                0 -- File
                                1 -- Directory
                                2 -- Symbolic Link
                                3 -- Character device
                                4 -- Block device
                                5 -- FIFO
                                6 -- Socket
  LARGE_INTEGER DevMajor;     Major device number if file type is device.
  LARGE_INTEGER DevMinor;     Minor device number if file type is device.
  LARGE_INTEGER UniqueId;     This is a server-assigned unique id for the
                                file. The client will typically map this onto
                                an inode number.  The scope of uniqueness is
                                the share.
  LARGE_INTEGER Permissions;  Standard UNIX file permissions
  LARGE_INTEGER Nlinks;       The number of directory entries that map to
                                this entry or number of hard links.
```

4.2.16.13.SMB_QUERY_FILE_UNIX_LINK

Used to retrieve destination file of a symbolic link

```
Data Block Encoding               Description
 ====================             ============
  STRING LinkDest;                  Destination for symbolic link
```

4.2.16.14.SMB_MAC_DT_GET_APPL

The Macintosh needs to be able to get an application name and its creator from a database. The Client sends a Trans2_Query_Path_Information call in which the name field is just ignored. The Client will send an info level that represents getting an application name with a structure that contains the File Creator and index. Where index has the following meaning.

- Index = 0; Get the application path from the database with the most current date.
- Index > 0; Use the index to find the application path from the database. e.g. index of 5 means get the fifth entries of this application name in the database.
- If no more entry return an error. The Server returns with a structure that contains the full path to the application and it's creator's data.
- Supporting the Desktop Database calls requires having a way to store information in a database. There are two kinds of information store in the database. Applications path that

is associated with an application signature. Icons are stored based on size, icon type, file creator, and file type.

Data Block Encoding	Description
ULONG FileCreator;	The application's signature. Always in big endian.
WORD Index;	

Response Field	Description
LARGE_INTEGER CreationTime;	The application's creation time NT date type
LONG FullPathLength;	Length field for Unicode
STRING FullPath;	If Unicode supported then Unicode string otherwise a ASCII string

4.2.16.15.SMB_MAC_DT_GET_ICON

The Macintosh needs to be able to get an icon from a database. The Client sends a Trans2_Query_Path_Information call in which the path name is ignored. The Client will send an info level that represents getting an icon with a structure that contains the Requested size of the icon, the Icon type, File Creator, and File Type. The Server returns with a structure that contains the actual size of the icon (must be less than requested length) and the icon bit map.

Data Block Encoding	Description
ULONG ReqCount;	Size of the icon being requested
ULONG FileCreator;	The application's signature. Always in big endian.
ULONG FileType;	The application's type. Always in Big Endian
WORD IconType;	The icon type. Always in Big Endian

Response Field	Description
UCHAR IconData[];	Icon data. Always in Big Endian

4.2.16.16.SMB_MAC_DT_GET_ICON_INFO

The Macintosh needs to be able to get an icon from a database. The Client sends a Trans2_Query_Path_Information call in which the path name is ignored. The Client will send an info level that represents getting an icon with a structure that contains the index and File Creator. The index allows the client to make repeated calls to the server gathering all icon stored by this file creator. The Server returns with a structure that contains the actual size of the icon (must be less than requested length) and the icon bit map, File Type, and Icon Type.

Data Block Encoding	Description
ULONG FileCreator;	The application's signature. Always in big endian.
ULONG Index;	

Response Field	Description
ULONG ActCount;	Size of the icon being requested
ULONG FileType;	The application's type. Always in Big Endian
WORD IconType;	The icon type. Always in Big Endian

4.2.16.17.Errors

```
ERRDOS codes
------------
ERRbadfile
ERRbadpath
ERRnoaccess
ERRnomem

ERRSRV codes
------------
ERRaccess
ERRinvdevice
ERRinvtid
ERRbaduid
```

4.2.17. TRANS2_QUERY_FILE_INFORMATION: Get File Attributes Given FID

This request is used to get information about a specific file or subdirectory given a handle to it.

```
Client Request             Value
===============            ======
 WordCount                  15
 MaxSetupCount              0
 SetupCount                 1
 Setup[0]                   TRANS2_QUERY_FILE_INFORMATION

Parameter Block Encoding   Description
=========================  ============
 USHORT Fid;                Handle of file for request
 USHORT InformationLevel;   Level of information requested
```

The available information levels, as well as the format of the response are identical to TRANS2_QUERY_PATH_INFORMATION.

4.2.18. TRANS2_SET_PATH_INFORMATION: Set File Attributes given Path

This request is used to set information about a specific file or subdirectory.

```
Client Request             Value
===============            ======
 WordCount                  15
 MaxSetupCount              0
 SetupCount                 1
 Setup[0]                   TRANS2_SET_PATH_INFORMATION
```

```
Parameter Block Encoding   Description
=========================  ============
 USHORT InformationLevel;   Level of information to set
 ULONG Reserved;            Must be zero
 STRING FileName;           File or directory name
```

The following Information Levels may be set:

InformationLevel Name	Value	Meaning
SMB_INFO_STANDARD	1	
SMB_INFO_QUERY_EA_SIZE	2	
SMB_INFO_QUERY_ALL_EAS	4	
SMB_SET_FILE_UNIX_BASIC	0x200	
SMB_SET_FILE_UNIX_LINK	0x201	
SMB_SET_FILE_UNIX_HLINK	0x203	

The response formats are:

4.2.18.1. SMB_INFO_STANDARD & SMB_INFO_QUERY_EA_SIZE

```
Parameter Block Encoding          Description
========================          ============
 USHORT Reserved                   0

Data Block Encoding               Description
====================              ============
 SMB_DATE CreationDate;            Date when file was created
 SMB_TIME CreationTime;            Time when file was created
 SMB_DATE LastAccessDate;          Date of last file access
 SMB_TIME LastAccessTime;          Time of last file access
 SMB_DATE LastWriteDate;           Date of last write to the file
 SMB_TIME LastWriteTime;           Time of last write to the file
 ULONG  DataSize;                  File Size
 ULONG AllocationSize;             Size of filesystem allocation
                                    unit
 USHORT Attributes;                File Attributes
 ULONG EaSize;                     Size of file's EA information
                                    (SMB_INFO_QUERY_EA_SIZE)
```

4.2.18.2. SMB_INFO_QUERY_ALL_EAS

```
Response Field           Value
==============           ======
 MaxDataCount             Length of FEAlist found (minimum value is 4)

Parameter Block
Encoding                  Description
================          ============
 USHORT EaErrorOffset;    Offset into EAList of EA error
```

```
Data Block Encoding       Description
====================      ============
 ULONG ListLength;         Length of the remaining data
 UCHAR EaList[];           The extended attributes list
```

4.2.18.3. SMB_SET_FILE_UNIX_BASIC

Used to set UNIX specific file attributes and create files

```
Data Block Encoding               Description
 ====================              ============
 LARGE_INTEGER EndOfFile;          File size
 LARGE_INTEGER NumOfBytes;          Number of file system bytes used to
                                    store file
 TIME LastStatusChange;            Last time the status of the file was
                                    changed. This is in DCE time.
 TIME LastAccessTime;              Time of last file access.  This is DCE
                                    time.
 TIME LastModificationTime;        Last modification time.  This is DCE
                                    time.
 LARGE_INTEGER Uid;                Numeric user id for the owner
 LARGE_INTEGER Gid;                Numeric group id of owner
 ULONG Type;                       Enumeration specifying the file type.
                                          0 -- File
                                          1 -- Directory
                                          2 -- Symbolic Link
                                          3 -- Character device
                                          4 -- Block device
                                          5 -- FIFO
                                          6 -- Socket
  LARGE_INTEGER DevMajor;           Major device number if file type is
                                    device
  LARGE_INTEGER DevMinor;           Minor device number if file type is
                                    device
  LARGE_INTEGER UniqueId;           This is a server-assigned unique id
                                     for the file. The client will
                                     typically map this onto an inode
                                     number.  The scop of uniqueness is
                                     the share
  LARGE_INTEGER Permissions;        Standard UNIX file permissions
  LARGE_INTEGER Nlinks;             The number of directory entries that
                                     map to this entry or number of hard
                                     links
```

4.2.18.4. SMB_SET_FILE_UNIX_LINK

Used to create symbolic link file.

```
Data Block Encoding               Description
 ====================              ============
  STRING LinkDest;                  Destination for symbolic link
```

4.2.18.5. SMB_SET_FILE_UNIX_HLINK

Used to create hard link file.

```
Data Block Encoding                Description
 ====================              ============
  STRING LinkDest;                   Destination for hard link
```

4.2.18.6. SMB_MAC_SET_FINDER_INFO

Parameter Block Encoding	Description
USHORT Reserved	0

Data Block Encoding	Description
WORD Type;	Type of action to take, described below
UCHAR FLAttrib;	Macintosh SetFLock if a 1 then the file is Macintosh locked
UCHAR Pad;	
LARGE_INTEGER CreationTime;	Time of file creation
LARGE_INTEGER LastWriteTime;	Time of file last modify
LARGE_INTEGER ChangeTime;	Time of file last change
ULONG ExtFileAttributes;	Extended file attributes
UCHAR FndrInfo1[16];	Information set by the finder. Described above in MacFindBothInfo structure
UCHAR FndrInfo2[16];	Information set by the finder. Described above in MacFindBothInfo structure

Listed below are the types of actions that the client may request with this Information Level:

SetCreateDate	0x0001	If this is set then set the create date of the file/folder
SetModDate	0x0002	If this is set then set the modify date of the file/folder
SetFLAttrib	0x0004	If this is set then set the Macintosh lock bit of the file/folder
FndrInfo1	0x0008	If this is set then set the first 16 bytes of finder info
FndrInfo2	0x0010	If this is set then set the second 16 bytes of finder info
SetHidden	0x0020	The Client is either setting or unsetting the hidden bit

4.2.18.7. SMB_MAC_DT_ADD_APPL

The Macintosh needs to be able to store an application name and its creator in a database. The Client sends a Trans2_Set_Path_Information call with the full path of the application in the path field. The Client sends an info level that represents adding an application name and creator to the database. The Client will pass the File Creator in the data message. The Server should just respond with no error if it was successful or an error if the operation failed

.

Parameter Block Encoding	Description
USHORT Reserved	0

Data Block Encoding	Description
ULONG FileCreator;	The application's signature. Always in big endian. The path name passed in this calls needs to be stored with this signature.

4.2.18.8. SMB_MAC_DT_REMOVE_APPL

The Macintosh needs to be able to remove an application name and its creator from a database. The Client sends a Trans2_Set_Path_Information call with the full path of the application in the path field. The Client will send an info level that represents removing an application name and creator from the database. The Client will pass the File Creator in the data message. The Server should just respond with no error if it was successful or an error it the operation failed.

Parameter Block Encoding	Description
USHORT Reserved	0

Data Block Encoding	Description
ULONG FileCreator;	The application's signature. Always in big endian. The path name passed in this calls needs to be removed with this signature.

4.2.18.9. SMB_MAC_DT_ADD_ICON

The Macintosh needs to be able to add an icon to a database. The Client sends a Trans2_Set_Path_Information call in which the path name is ignored. The Client will send an info level that represents setting an icon with a structure that contains the icon data, icon size, icon type, the file type, and file creator. The Server returns only if the call was successful or not.

Parameter Block Encoding	Description
USHORT Reserved	0

Data Block Encoding	Description
ULONG IconSize;	Size of the icon in bytes.
ULONG FileCreator;	The application's signature. Always in big endian.
ULONG FileType;	The application's type. Always in big endian.
WORD IconType;	The icon type. Always in big endian.
UCHAR IconData[];	Icon data,

4.2.18.10. Errors

```
ERRDOS codes
------------
ERRbadfile
ERRbadpath
ERRnoaccess
ERRnomem
ERRbadaccess
ERRbadshare
```

```
ERRSRV codes
------------
ERRaccess
ERRinvdevice
ERRinvtid
ERRbaduid
```

4.2.19. TRANS2_SET_FILE_INFORMATION: Set File Attributes Given FID

This request is used to set information about a specific file or subdirectory given a handle to the file or subdirectory.

```
Client Request             Value
==============             ======
 WordCount                  15
 MaxSetupCount              0
 SetupCount                 1
 Setup[0]                   TRANS2_SET_FILE_INFORMATION

Parameter Block Encoding   Description
=========================  ============
 USHORT Fid;                Handle of file for request
 USHORT InformationLevel;   Level of information requested
 USHORT Reserved;           Ignored by the server
```

The following InformationLevels may be set:

InformationLevel Name	Value	Meaning
SMB_INFO_STANDARD	1	
SMB_INFO_QUERY_EA_SIZE	2	
SMB_SET_FILE_BASIC_INFO	0x101	
SMB_SET_FILE_DISPOSITION_INFO	0x102	
SMB_SET_FILE_ALLOCATION_INFO	0x103	
SMB_SET_FILE_END_OF_FILE_INFO	0x104	
SMB_SET_FILE_UNIX_BASIC	0x200	
SMB_SET_FILE_UNIX_LINK	0x201	
SMB_SET_FILE_UNIX_HLINK	0x203	

The two levels below 0x101 and the three levels 0x200, 0x201, and 0x202 are as described in the NT_SET_PATH_INFORMATION transaction. The requested information is placed in the Data portion of the transaction response. For the information levels greater than 0x100 and below 0x200, the transaction response has 1 parameter word, which should be ignored by the client.

4.2.19.1. SMB_FILE_BASIC_INFO

```
Data Block Encoding              Description
 ====================             ============
  TIME CreationTime;             Time when file was created
  TIME LastAccessTime;           Time of last file access
  TIME LastWriteTime;            Time of last write to the file
  TIME ChangeTime;               Time when file was last changed
  ULONG Attributes;              File Attributes
```

The valid file attributes are listed in section 4.2.15.4 SMB_QUERY_FILE_BASIC_INFO:

4.2.19.2. SMB_FILE_DISPOSITION_INFO

Response Field	Value
BOOLEAN FileIsDeleted;	A boolean which is TRUE if the file is marked for deletion

4.2.19.3. SMB_FILE_ALLOCATION_INFO

Response Field	Value
LARGE_INTEGER	File Allocation size in number of bytes

4.2.19.4. SMB_FILE_END_OF_FILE_INFO

Response Field	Value
LARGE_INTEGER	The total number of bytes that need to be traversed from the beginning of the file in order to locate the end of the file

4.2.19.5. Errors

ERRDOS codes

ERRbadfile
ERRbadpath
ERRnoaccess
ERRnomem
ERRbadaccess
ERRbadshare

ERRSRV codes

ERRaccess
ERRinvdevice
ERRinvtid
ERRbaduid

4.3. Directory Requests

4.3.1. TRANS2_CREATE_DIRECTORY: Create Directory (with optional EAs)

This requests the server to create a directory relative to Tid in the SMB header, optionally assigning extended attributes to it.

Client Request	Value
WordCount	15
MaxSetupCount	0
SetupCount	1
Setup[0]	TRANS2_CREATE_DIRECTORY

```
Parameter Block Encoding   Description
=========================  ============
 ULONG Reserved;            Reserved--must be zero
 STRING Name[];             Directory name to create
 UCHAR Data[];              Optional FEAList for the new directory

Response Parameter Block   Description
=========================  ============
 USHORT EaErrorOffset       Offset into FEAList of first error which
                             occurred while setting Eas
```

4.3.1.1. Errors

```
 ERRDOS codes
 ------------
 ERRbadfile
 ERRbadpath
 ERRnoaccess
 ERRnomem
 ERRbadaccess
 ERRfileexists
 ERRquota

 ERRSRV codes
 ------------
 ERRaccess
 ERRinvdevice
 ERRinvtid
 ERRbaduid
```

4.3.2. DELETE_DIRECTORY: Delete Directory

The delete directory message is sent to delete an empty directory. The appropriate Tid and additional pathname are passed. The directory must be empty for it to be deleted.

```
Client Request             Description
===============            ============
 UCHAR WordCount;           Count of parameter words = 0
 USHORT ByteCount;          Count of data bytes; min = 2
 UCHAR BufferFormat;        0x04
 STRING DirectoryName[];    Directory name
```

The directory to be deleted cannot be the root of the share specified by Tid.

```
Server Response            Description
===============            ============
 UCHAR WordCount;           Count of parameter words = 0
 USHORT ByteCount;          Count of data bytes = 0
```

4.3.2.1. Errors

```
 ERRDOS codes
 ------------
 ERRbadfile
 ERRbadpath
 ERRnoaccess
```

```
ERRnomem
ERRbadaccess
ERRfileexists

ERRSRV codes
------------
ERRaccess
ERRinvdevice
ERRinvtid
ERRbaduid
```

4.3.3. CHECK_DIRECTORY: Check Directory

This SMB is used to verify that a path exists and is a directory. No error is returned if the given path exists and the client has read access to it. When the path turns out to specify a file (non-directory) then STATUS_NOT_A_DIRECTORY is returned. Client machines which maintain a concept of a "working directory" will find this useful to verify the validity of a "change working directory" command. Note that the servers do NOT have a concept of working directory for a particular client. The client must always supply full pathnames relative to the Tid in the SMB header.

```
Client Request              Description
==============              ============
 UCHAR WordCount;           Count of parameter words = 0
 USHORT ByteCount;          Count of data bytes; min = 2
 UCHAR BufferFormat;        0x04
 STRING DirectoryPath[];    Directory path

Server Response             Description
===============             ============
 UCHAR WordCount;           Count of parameter words = 0
 USHORT ByteCount;          Count of data bytes = 0
```

DOS clients, in particular, depend on the SMB_ERR_BAD_PATH return code if the directory is not found.

4.3.3.1. Errors

```
ERRDOS/ERRbadfile
ERRDOS/ERRbadpath
ERRDOS/ERRnoaccess
ERRHRD/ERRdata
ERRSRV/ERRinvid
ERRSRV/ERRbaduid
ERRSRV/ERRaccess
```

4.3.4. TRANS2_FIND_FIRST2: Search Directory using Wildcards

```
Client Request              Value
==============              ======
 WordCount                  15
 TotalDataCount             Total size of extended attribute list
 SetupCount                 1
 Setup[0]                   TRANS2_FIND_FIRST2
```

```
Parameter Block Encoding     Description
=========================    ============
 USHORT SearchAttributes;
 USHORT SearchCount;          Maximum number of entries to return
 USHORT Flags;                Additional information:
                               Bit 0 - close search after this request
                               Bit 1 - close search if end of search
                                reached
                               Bit 2 - return resume keys for each
                                entry found
                               Bit 3 - continue search from previous
                                ending place
                               Bit 4 - find with backup intent
 USHORT InformationLevel;     See below
 ULONG SearchStorageType;
 STRING FileName;             Pattern for the search
 UCHAR Data[TotalDataCount];  FEAList if InformationLevel is
                               QUERY_EAS_FROM_LIST

Response Parameter Block     Description
=========================    ============
 USHORT Sid;                  Search handle
 USHORT SearchCount;          Number of entries returned
 USHORT EndOfSearch;          Was last entry returned?
 USHORT EaErrorOffset;        Offset into EA list if EA error
 USHORT LastNameOffset;       Offset into Data[] holding the file name of
                               the last entry, if server needs it to resume
                               search; else 0
 UCHAR Data[TotalDataCount];  Level dependent info about the matches
                               found in the search
```

This request allows the client to search for the file(s) which match the file specification. The search can be continued if necessary with TRANS2_FIND_NEXT2. There are numerous levels of information which may be obtained for the returned files, the desired level is specified in the InformationLevel field of the request. The following values can be specified for InformationLevel:

InformationLevel Name	Value	Meaning
SMB_INFO_STANDARD	1	
SMB_INFO_QUERY_EA_SIZE	2	
SMB_INFO_QUERY_EAS_FROM_LIST	3	
SMB_FIND_FILE_DIRECTORY_INFO	0x101	
SMB_FIND_FILE_FULL_DIRECTORY_INFO	0x102	
SMB_FIND_FILE_NAMES_INFO	0x103	
SMB_FIND_FILE_BOTH_DIRECTORY_INFO	0x104	
SMB_FIND_FILE_UNIX	0x202	

The following sections detail the data returned for each InformationLevel. The requested information is placed in the Data portion of the transaction response. Note: a client which does not support long names can only request SMB_INFO_STANDARD.

The search Id is the Search Handle returned back from the server on the FindFirst response which can be used on the FindNext request so that the full path can be avoided. Search Handle is session wide. The server doesn't care what process uses it on the client.

A four-byte resume key precedes each data item (described below). The return of resume keys is dependent upon setting the flag SMB_FIND_RETURN_RESUME_KEYS in the FLAGS of the REQ_FIND_NEXT2 packet. The resume key tells the server where to resume the operation on the FindNext request in order to avoid duplicate entries. The contents of the resume key are opaque to the client.

If the search doesn't find any names, the server should return either `STATUS_NO_SUCH_FILE` or the corresponding error code `ERROR_FILE_NOT_FOUND.`

4.3.4.1. SMB_INFO_STANDARD

```
Response Field                  Description
===============                 ============
 SMB_DATE CreationDate;          Date when file was created
 SMB_TIME CreationTime;          Time when file was created
 SMB_DATE LastAccessDate;        Date of last file access
 SMB_TIME LastAccessTime;        Time of last file access
 SMB_DATE LastWriteDate;         Date of last write to the file
 SMB_TIME LastWriteTime;         Time of last write to the file
 ULONG  DataSize;                File Size
 ULONG AllocationSize;           Size of filesystem allocation unit
 USHORT Attributes;              File Attributes
 UCHAR FileNameLength;           Length of filename in bytes
 STRING FileName;                Name of found file
```

4.3.4.2. SMB_INFO_QUERY_EA_SIZE

```
Response Field                  Description
===============                 ============
 SMB_DATE CreationDate;          Date when file was created
 SMB_TIME CreationTime;          Time when file was created
 SMB_DATE LastAccessDate;        Date of last file access
 SMB_TIME LastAccessTime;        Time of last file access
 SMB_DATE LastWriteDate;         Date of last write to the file
 SMB_TIME LastWriteTime;         Time of last write to the file
 ULONG DataSize;                 File Size
 ULONG AllocationSize;           Size of filesystem allocation unit
 USHORT Attributes;              File Attributes
 ULONG EaSize;                   Size of file's EA information
 UCHAR FileNameLength;           Length of filename in bytes
 STRING FileName;                Name of found file
```

4.3.4.3. SMB_INFO_QUERY_EAS_FROM_LIST

This request returns the same information as SMB_INFO_QUERY_EA_SIZE, but only for files which have an EA list which match the EA information in the Data part of the request.

4.3.4.4. SMB_FIND_FILE_DIRECTORY_INFO

```
Response Field                  Description
===============                 ==================================
 ULONG NextEntryOffset;          Offset from this structure to
```

```
                                    the beginning of the next one
 ULONG FileIndex;
 TIME  CreationTime;           File creation time
 TIME  LastAccessTime;         Last access time for the file
 TIME  LastWriteTime;          Last write time for the file
 TIME  ChangeTime;             Last attribute change time for the file
 LARGE_INTEGER EndOfFile;         File size
 LARGE_INTEGER AllocationSize;    Size of filesystem allocation
                                   information
 ULONG ExtFileAttributes;         Extended file attributes (see
                                   Section 3.12)
 ULONG FileNameLength;            Length of filename in bytes
 STRING FileName;                 Name of the file
```

4.3.4.5. SMB_FIND_FILE_FULL_DIRECTORY_INFO

```
Response Field                  Description
===============                 ============
 ULONG NextEntryOffset;          Offset from this structure to
                                  the beginning of the next one
 ULONG FileIndex;
 TIME  CreationTime;           File creation time
 TIME  LastAccessTime;         Last access time for the file
 TIME  LastWriteTime;          Last write time for the file
 TIME  ChangeTime;             Last attribute change time for the file
 LARGE_INTEGER EndOfFile;         File size
 LARGE_INTEGER AllocationSize;    Size of filesystem allocation
                                   information
 ULONG ExtFileAttributes;         Extended file attributes (see
                                   Section 3.12)
 ULONG FileNameLength;            Length of filename in bytes
 ULONG EaSize;                    Size of file's extended attributes
 STRING FileName;                 Name of the file
```

4.3.4.6. SMB_FIND_FILE_BOTH_DIRECTORY_INFO

```
Response Field                  Description
===============                 ============
 ULONG NextEntryOffset;          Offset from this structure to
                                  the beginning of the next one
 ULONG FileIndex;
 TIME  CreationTime;           File creation time
 TIME  LastAccessTime;         Last access time for the file
 TIME  LastWriteTime;          Last write time for the file
 TIME  ChangeTime;             Last attribute change time for the file
 LARGE_INTEGER EndOfFile;         File size
 LARGE_INTEGER AllocationSize;    Size of filesystem allocation
                                   information
 ULONG ExtFileAttributes;         Extended file attributes (see
                                   Section 3.12)
 ULONG FileNameLength;            Length of FileName in bytes
 ULONG EaSize;                    Size of file's extended attributes
 UCHAR ShortNameLength;           Length of file's short name in
                                   bytes
 UCHAR Reserved;
```

```
WCHAR ShortName[12];            File's 8.3 conformant name in Unicode

STRING FileName;                File's full length name
```

4.3.4.7. SMB_FIND_FILE_NAMES_INFO

```
Response Field                  Description
==============                  ============
ULONG NextEntryOffset;          Offset from this structure to
                                 the beginning of the next one
ULONG FileIndex;
ULONG FileNameLength;           Length of FileName in bytes
STRING FileName;                File's full length name
```

4.3.4.8. SMB_FIND_FILE_UNIX

Used to return UNIX attribute information in a file search response

```
Data Block Encoding          Description
====================         ============
ULONG NextEntryOffset;       Offset from this structure to the beginning
                               of the next one
ULONG ResumeKey;             Used for continuing search
LARGE_INTEGER EndOfFile;     File size
LARGE_INTEGER NumOfBytes     Number of file system bytes used to store
                              file
TIME LastStatusChange;       Last time the status of the file was changed.
                              This is in DCE time.
TIME LastAccessTime;         Time of last file access.  This is DCE time.
TIME LastModificationTime;   Last modification time.  This is DCE time.
LARGE_INTEGER Uid;           Numeric user id for the owner
LARGE_INTEGER Gid;           Numeric group id of owner
ULONG Type;                  Enumeration specifying the file type.
                                     0 -- File
                                     1 -- Directory
                                     2 -- Symbolic Link
                                     3 -- Character device
                                     4 -- Block device
                                     5 -- FIFO
                                     6 -- Socket
LARGE_INTEGER DevMajor;      Major device number if file type is device
LARGE_INTEGER DevMinor;      Minor device number if file type is device
LARGE_INTEGER UniqueId;      This is a server-assigned unique id for the
                               file. The client will typically map this onto
                               an inode number.  The scop of uniqueness is
                               the share
LARGE_INTEGER Permissions;   Standard UNIX file permissions
LARGE_INTEGER Nlinks;        The number of directory entries that map to
                               this entry or number of hard links
STRING Name;                 Case-preserved alternative filename
```

4.3.4.9. SMB_ FINDBOTH_ MAC_HFS_INFO

Response Field	Description
ULONG NextEntryOffset;	Offset from this structure to beginning of next one
ULONG FileIndex;	
LARGE_INTEGER CreationTime;	file creation time
LARGE_INTEGER LastWriteTime;	last write time
LARGE_INTEGER ChangeTime;	last attribute change time
LARGE_INTEGER EndOfFile;	Data stream file size
LARGE_INTEGER EndOfFile_R;	Resource stream file size
LARGE_INTEGER AllocationSize;	Data stream size of file system allocation information
LARGE_INTEGER AllocationSize_R;	Resource stream size of file system allocation information
ULONG ExtFileAttributes;	Extended file attributes
UCHAR FLAttrib;	Macintosh SetFLock if a 1 then the file is locked.
UCHAR Pad;	
UWORD DrNmFls;	If a directory the number of items in that directory otherwise ignored.
ULONG AccessCntrl;	Ignored unless SUPPORT_MAC_ACCESS_CNTRL is set.
UCHAR FndrInfo[32];	FndrInfo[32]; Information used by the finder that is always in Big Endian. Bytes 0-3 File Type If a file default to 'TEXT' otherwise default to zero Bytes 4-7 File Creator If a file default to 'dosa' otherwise default to zero Bytes 8-9 a UWORD flags field If hidden item set this UWORD to 0x4000 else defaults to zero All other bytes should default to zero and are only changeable by the Macintosh
ULONG FileNameLength;	Length of Filename in bytes
UCHAR ShortNameLength;	Length of file's short name in bytes
UCHAR Reserved	
WCHAR ShortName[12];	File's 8.3 conformant name in Unicode
STRING Filename;	Files full length name
LONG UniqueFileID;	Unique file or directory identifier - only supported if the SUPPORT_MAC_UNIQUE_IDS bit is set in the MacSupportFlags.

4.3.4.10. Errors

```
ERRDOS codes
------------
ERRbadpath
ERRnoaccess
ERRnomem

ERRSRV codes
------------
ERRaccess
ERRinvdevice
ERRinvtid
ERRbaduid
```

4.3.5. TRANS2_FIND_NEXT2: Resume Directory Search Using Wildcards

This request resumes a search which was begun with a previous TRANS2_FIND_FIRST2 request.

```
Client Request                  Value
==============                  ======
WordCount                       15
SetupCount                      1
Setup[0]                        TRANS2_FIND_NEXT2

Parameter Block Encoding        Description
========================        ============
USHORT Sid;                     Search handle
USHORT SearchCount;             Maximum number of entries to return
USHORT InformationLevel;        Levels described in
                                  TRANS2_FIND_FIRST2 request
ULONG ResumeKey;                Value returned by previous find2 call
USHORT Flags;                   Additional information: bit set-
                          0 - close search after this request
                          1 - close search if end of search reached
                          2 - return resume keys for each entry found
                          3 - resume/continue from previous ending place
                          4 - find with backup intent
STRING FileName;                Resume file name
```

Sid is the value returned by a previous successful TRANS2_FIND_FIRST2 call. If Bit3 of Flags is set, then FileName may be the NULL string, since the search is continued from the previous TRANS2_FIND request. Otherwise, FileName must not be more than 256 characters long.

```
Response Field                  Description
==============                  ============
USHORT SearchCount;             Number of entries returned
USHORT EndOfSearch;             Was last entry returned?
USHORT EaErrorOffset;           Offset into EA list if EA error
USHORT LastNameOffset;          Offset into Data[] holding the file name
                                 of the last entry, if server needs it
                                 to resume search; else 0
UCHAR Data[TotalDataCount];     Level dependent info about the
                                 matches found in the search
```

4.3.5.1. Errors

```
ERRDOS codes
------------
ERRnomem

ERRSRV codes
------------
ERRinvtid
ERRbaduid
```

4.3.6. FIND_CLOSE2: Close Directory Search

This SMB closes a search started by the TRANS2_FIND_FIRST2 transaction request.

```
Client Request                  Description
==============                  ============
 UCHAR WordCount;                Count of parameter words = 1
 USHORT Sid;                     Find handle
 USHORT ByteCount;               Count of data bytes = 0

Server Response                 Description
===============                 ============
 UCHAR WordCount;                Count of parameter words = 0
 USHORT ByteCount;               Count of data bytes = 0
```

4.3.6.1. Errors

```
ERRDOS/ERRbadfid
ERRSRV/ERRinvid
ERRSRV/ERRaccess
```

4.3.7. NT_TRANSACT_NOTIFY_CHANGE: Request Change Notification

```
 Client Setup Words             Description
 ==================             ============
  ULONG CompletionFilter;        Specifies operation to monitor
  USHORT Fid;                    Fid of directory to monitor
  BOOLEAN WatchTree;             TRUE = Watch all subdirectories too
UCHAR Reserved;                 MUST BE ZERO
```

This command notifies the client when the directory specified by Fid is modified. It also returns the name(s) of the file(s) that changed. The command completes once the directory has been modified based on the supplied CompletionFilter. The command is a "single shot" and therefore needs to be reissued to watch for more directory changes.

A directory file must be opened before this command may be used. Once the directory is open, this command may be used to begin watching files and subdirectories in the specified directory for changes. The first time the command is issued, the MaxParameterCount field in the transact header determines the size of the buffer that will be used at the server to buffer directory change information between issuances of the notify change commands.

When a change that is in the CompletionFilter is made to the directory, the command completes. The names of the files that have changed since the last time the command was issued are returned to the client. The ParameterCount field of the response indicates the number of bytes that are being returned. If too many files have changed since the last time the command was

issued, then zero bytes are returned and the NTSTATUS code STATUS_NOTIFY_ENUM_DIR (0x0000010C) is returned in the Status field of the response.

The CompletionFilter is a mask created as the sum of any of the following flags:

```
FILE_NOTIFY_CHANGE_FILE_NAME        0x00000001
FILE_NOTIFY_CHANGE_DIR_NAME         0x00000002
FILE_NOTIFY_CHANGE_NAME             0x00000003
FILE_NOTIFY_CHANGE_ATTRIBUTES       0x00000004
FILE_NOTIFY_CHANGE_SIZE             0x00000008
FILE_NOTIFY_CHANGE_LAST_WRITE       0x00000010
FILE_NOTIFY_CHANGE_LAST_ACCESS      0x00000020
FILE_NOTIFY_CHANGE_CREATION         0x00000040
FILE_NOTIFY_CHANGE_EA               0x00000080
FILE_NOTIFY_CHANGE_SECURITY         0x00000100
FILE_NOTIFY_CHANGE_STREAM_NAME      0x00000200
FILE_NOTIFY_CHANGE_STREAM_SIZE      0x00000400
FILE_NOTIFY_CHANGE_STREAM_WRITE     0x00000800

Server Response                     Description
================                    ============
 ParameterCount                      # of bytes of change data
 Parameters[ParameterCount ]         FILE_NOTIFY_INFORMATION
                                      Structures
```

The response contains FILE_NOTIFY_INFORMATION structures, as defined below. The NextEntryOffset field of the structure specifies the offset, in bytes, from the start of the current entry to the next entry in the list. If this is the last entry in the list, this field is zero. Each entry in the list must be longword aligned, so NextEntryOffset must be a multiple of four.

```
typedef struct {
    ULONG NextEntryOffset;
    ULONG Action;
    ULONG FileNameLength;
    WCHAR FileName[1];
} FILE_NOTIFY_INFORMATION;
```

Where Action describes what happened to the file named FileName:

```
FILE_ACTION_ADDED            0x00000001
FILE_ACTION_REMOVED          0x00000002
FILE_ACTION_MODIFIED         0x00000003
FILE_ACTION_RENAMED_OLD_NAME 0x00000004
FILE_ACTION_RENAMED_NEW_NAME 0x00000005
FILE_ACTION_ADDED_STREAM     0x00000006
FILE_ACTION_REMOVED_STREAM   0x00000007
FILE_ACTION_MODIFIED_STREAM  0x00000008
```

The client waits on the response after it sends the notify change request. If the client wants to discard the request, it can send NT_CANCEL to the server which should return STATUS_CANCELED. The server can reject the request with STATUS_NOT_IMPLEMENTED.

4.3.7.1. Errors

```
ERRDOS codes
------------
ERRbadpath
ERRnoaccess
ERRnomem

ERRSRV codes
------------
ERRaccess
ERRinvdevice
ERRinvtid
ERRbaduid
ERRSRV/ERROR_NOTIFY_ENUM_DIR
```

4.4. DFS Operations

4.4.1. TRANS2_GET_DFS_REFERRAL: Retrieve Distributed Filesystem Referral

The client sends this request to ask the server to convert RequestFilename into an alternate name for this file. This request can be sent to the server if the server response to the NEGOTIATE SMB included the CAP_DFS capability. The TID of the request must be IPC$. Bit15 of Flags2 in the SMB header must be set, indicating this is a UNICODE request.

Client Request	Description
WordCount	15
TotalDataCount	0
SetupCount	1
Setup[0]	TRANS2_GET_DFS_REFERRAL

Parameter Block Encoding	Description
USHORT MaxReferralLevel;	Latest referral version number understood
WCHAR RequestFileName[];	DFS name of file for which referral is sought

Response Data Block	Description
USHORT PathConsumed;	Number of RequestFilename bytes consumed by the server
USHORT NumberOfReferrals;	Number of referrals contained in this response
USHORT Flags;	Bit0 - The servers in Referrals are capable of fielding TRANS2_GET_DFS_REFERRAL. Bit1 - The servers in Referrals should hold the storage for the requested file
REFERRAL_LIST Referrals[];	Set of referrals for this file
UNICODESTRING Strings;	Used to hold the strings pointed to by Version 2 Referrals in REFERRALS

The server response is a list of Referrals which inform the client where it should resubmit the request to obtain access to the file. PathConsumed in the response indicates to the client how many characters of RequestFilename have been consumed by the server. When the client chooses one of the referrals to use for file access, the client may need to strip the leading PathConsumed characters from the front of RequestFileName before submitting the name to the target server. Whether or not the pathname should be trimmed is indicated by the individual referral as detailed below.

Flags indicates how this referral should be treated. If bit0 is clear, any entity in the Referrals list holds the storage for RequestFileName. If bit0 is set, any entity in the Referrals list has further referral information for RequestFilename - a TRANS2_GET_DFS_REFERRAL request should be sent to an entity in the Referrals list for further resolution.

The format of an individual referral contains version and length information allowing the client to skip referrals it does not understand. MaxReferralLevel indicates to the server the latest version of referral which the client can digest. Since each referral has a uniform element, MaxReferralLevel is advisory only. Each element in Referrals has this envelope:

```
REFERRAL_LIST Element
=====================
 USHORT VersionNumber;        Version of this referral element
 USHORT ReferralSize;         Size of this referral element
```

The following referral element versions are defined:

```
Version 1 Referral Element Format
=================================
 USHORT ServerType;           Type of Node handling referral:
                               0 - Don't know
                               1 - SMB Server
                               2 - Netware Server
                               3 - Domain
 USHORT ReferralFlags;        Flags which describe this referral:
                               01 - Strip off PathConsumed characters
                                before submitting RequestFileName to Node
 UNICODESTRING Node;          Name of entity to visit next

Version 2 Referral Element Format
=================================
 USHORT ServerType;           Type of Node handling referral:
                               0 - Don't know
                               1 - SMB Server
                               2 - Netware Server
                               3 - Domain
 USHORT ReferralFlags;        Flags which describe this referral:
                               01 - Strip off PathConsumed characters
                                before submitting RequestFileName to Node
 ULONG Proximity;             A hint describing the proximity of this
                               server to the client. 0 indicates the
                               closest, higher numbers indicate
                               increasingly "distant" servers. The
                               number is only relevant within the
                               context of the servers listed in this
                               particular SMB.
 ULONG TimeToLive;            Number of seconds for which the client
                               can cache this referral.
```

```
USHORT DfsPathOffset;         Offset, in bytes from the beginning of
                               this referral, of the DFS Path that
                               matched PathConsumed bytes of the
                               RequestFileName.
USHORT                        Offset, in bytes from the beginning of
  DfsAlternatePathOffset;      this referral, of an alternate name
                               (8.3 format) of the DFS Path that
                               matched PathConsumed bytes of the
                               RequestFileName.
USHORT NetworkAddressOffset; Offset, in bytes from the beginning of
                               this referral, of the entity to visit next.
```

The CIFS protocol imposes no referral selection policy.

4.4.1.1. Errors

```
ERRDOS codes
------------
ERRnoaccess
ERRnomem

ERRSRV codes
------------
ERRaccess
ERRinvdevice
ERRinvtid
ERRbaduid
```

4.4.2. TRANS2_REPORT_DFS_INCONSISTENCY: Inform a server about DFS Error

As part of the Distributed Name Resolution algorithm, a DFS client may discover a knowledge inconsistency between the referral server (i.e., the server that handed out a referral), and the storage server (i.e., the server to which the client was redirected by the referral server). When such an inconsistency is discovered, the DFS client optionally sends this SMB to the referral server, allowing the referral server to take corrective action.

```
Client Request                  Description
===============                 ============
 WordCount                       15
 MaxParameterCount               0
 SetupCount                      1
 Setup[0]                        TRANS2_REPORT_DFS_INCONSISTENCY

Parameter Block Encoding        Description
========================        ============
 UNICODESTRING RequestFileName;  DFS Name of file for which
                                  referral is sought
```

The data part of this request contains the referral element (Version 1 format only) believed to be in error. These are encoded as described in the TRANS2_GET_DFS_REFERRAL response. If the server returns success, the client can resubmit the TRANS2_GET_DFS_REFERRAL request to this server to get a new referral. It is not mandatory for the DFS knowledge to be automatically repaired - the client must be prepared to receive further errant referrals and must not wind up looping between this request and the TRANS2_GET_DFS_REFERRAL request.

Bit15 of Flags2 in the SMB header must be set, indicating this is a UNICODE request.

4.4.2.1. Errors

```
ERRDOS codes
------------
ERRnoaccess
ERRnomem

ERRSRV codes
------------
ERRaccess
ERRinvdevice
ERRinvtid
ERRbaduid
```

4.5. Miscellaneous Operations

4.5.1. NT_TRANSACT_IOCTL

This command allows device and file system control functions to be transferred transparently from client to server.

```
Setup Words Encoding        Description
====================        ============
 ULONG FunctionCode;         NT device or file system control code
 USHORT Fid;                 Handle for i/o or file system control,
                                     unless BIT0 of ISFLAGS is set
 BOOLEAN IsFsctl;            Indicates whether the command is for device
                              (FALSE) or a file system control (TRUE)
 UCHAR IsFlags;              BIT0 - command is to be applied to share
                              root handle.  Share must be a DFS share.

Data Block Encoding         Description
===================         ============
 UCHAR Data[                 Passed to the Fsctl or Ioctl
     TotalDataCount];

Server Response             Description
================            ============
 SetupCount                  1
 Setup[0]                    Length of information returned by
                              i/o or file system control
 DataCount                   Length of information returned by
                              i/o or file system control
 Data[DataCount]             The results of the i/o or file system
                              control
```

4.5.1.1. Errors

```
ERRDOS codes
------------
ERRnoaccess
ERRnomem
```

```
ERRSRV codes
------------
ERRaccess
ERRinvdevice
ERRinvtid
ERRbaduid
```

4.5.2. NT_TRANSACT_QUERY_SECURITY_DESC

This command allows the client to retrieve the security descriptor on a file.

```
Client Parameter Block      Description
======================      ============
 USHORT Fid;                 FID of target
 USHORT Reserved;            MUST BE ZERO
 ULONG SecurityInformation;  Fields of descriptor to get
```

NtQuerySecurityObject() is called, requesting SecurityInformation. The result of the call is returned to the client in the Data part of the transaction response.

4.5.2.1. Errors

```
ERRDOS codes
------------
ERRnoaccess
ERRnomem
ERRbadaccess

ERRSRV codes
------------
ERRaccess
ERRinvdevice
ERRinvtid
ERRbaduid
```

4.5.3. NT_TRANSACT_SET_SECURITY_DESC

This command allows the client to change the security descriptor on a file.

```
Client Parameter Block Encoding    Description
===============================    ============
 USHORT Fid;                        FID of target
 USHORT Reserved;                   MUST BE ZERO
 ULONG SecurityInformation;         Fields of Security Descriptor to set

Data Block Encoding               Description
===================               ============
 Data[TotalDataCount]              Security Descriptor information
```

Data is passed directly to NtSetSecurityObject(), with SecurityInformation describing which information to set. The transaction response contains no parameters or data.

4.5.3.1. Errors

```
ERRDOS codes
------------
ERRnoaccess
ERRnomem
ERRbadaccess
ERRbadshare

ERRSRV codes
------------
ERRaccess
ERRinvdevice
ERRinvtid
ERRbaduid
```

5. SMB Symbolic Constants

5.1. SMB Command Codes

The following values have been assigned for the SMB Commands.

```
SMB_COM_CREATE_DIRECTORY        0x00
SMB_COM_DELETE_DIRECTORY        0x01
SMB_COM_OPEN                    0x02
SMB_COM_CREATE                  0x03
SMB_COM_CLOSE                   0x04
SMB_COM_FLUSH                   0x05
SMB_COM_DELETE                  0x06
SMB_COM_RENAME                  0x07
SMB_COM_QUERY_INFORMATION       0x08
SMB_COM_SET_INFORMATION         0x09
SMB_COM_READ                    0x0A
SMB_COM_WRITE                   0x0B
SMB_COM_LOCK_BYTE_RANGE         0x0C
SMB_COM_UNLOCK_BYTE_RANGE       0x0D
SMB_COM_CREATE_TEMPORARY        0x0E
SMB_COM_CREATE_NEW              0x0F
SMB_COM_CHECK_DIRECTORY         0x10
SMB_COM_PROCESS_EXIT            0x11
SMB_COM_SEEK                    0x12
SMB_COM_LOCK_AND_READ           0x13
SMB_COM_WRITE_AND_UNLOCK        0x14
SMB_COM_READ_RAW                0x1A
SMB_COM_READ_MPX                0x1B
SMB_COM_READ_MPX_SECONDARY      0x1C
SMB_COM_WRITE_RAW               0x1D
SMB_COM_WRITE_MPX               0x1E
SMB_COM_WRITE_MPX_SECONDARY     0x1F
SMB_COM_WRITE_COMPLETE          0x20
SMB_COM_QUERY_SERVER            0x21
SMB_COM_SET_INFORMATION2        0x22
SMB_COM_QUERY_INFORMATION2      0x23
SMB_COM_LOCKING_ANDX            0x24
SMB_COM_TRANSACTION             0x25
SMB_COM_TRANSACTION_SECONDARY   0x26
SMB_COM_IOCTL                   0x27
SMB_COM_IOCTL_SECONDARY         0x28
SMB_COM_COPY                    0x29
SMB_COM_MOVE                    0x2A
SMB_COM_ECHO                    0x2B
SMB_COM_WRITE_AND_CLOSE         0x2C
SMB_COM_OPEN_ANDX               0x2D
SMB_COM_READ_ANDX               0x2E
SMB_COM_WRITE_ANDX              0x2F
SMB_COM_NEW_FILE_SIZE           0x30
SMB_COM_CLOSE_AND_TREE_DISC     0x31
SMB_COM_TRANSACTION2            0x32
SMB_COM_TRANSACTION2_SECONDARY  0x33
SMB_COM_FIND_CLOSE2             0x34
```

```
SMB_COM_FIND_NOTIFY_CLOSE       0x35
/* Used by Xenix/Unix 0x60 - 0x6E */
SMB_COM_TREE_CONNECT            0x70
SMB_COM_TREE_DISCONNECT         0x71
SMB_COM_NEGOTIATE               0x72
SMB_COM_SESSION_SETUP_ANDX      0x73
SMB_COM_LOGOFF_ANDX             0x74
SMB_COM_TREE_CONNECT_ANDX       0x75
SMB_COM_QUERY_INFORMATION_DISK  0x80
SMB_COM_SEARCH                  0x81
SMB_COM_FIND                    0x82
SMB_COM_FIND_UNIQUE             0x83
SMB_COM_FIND_CLOSE              0x84
SMB_COM_NT_TRANSACT             0xA0
SMB_COM_NT_TRANSACT_SECONDARY   0xA1
SMB_COM_NT_CREATE_ANDX          0xA2
SMB_COM_NT_CANCEL               0xA4
SMB_COM_NT_RENAME               0xA5
SMB_COM_OPEN_PRINT_FILE         0xC0
SMB_COM_WRITE_PRINT_FILE        0xC1
SMB_COM_CLOSE_PRINT_FILE        0xC2
SMB_COM_GET_PRINT_QUEUE         0xC3
SMB_COM_READ_BULK               0xD8
SMB_COM_WRITE_BULK              0xD9
SMB_COM_WRITE_BULK_DATA         0xDA
```

5.2. SMB_COM_TRANSACTION2 Subcommand codes

The subcommand code for SMB_COM_TRANSACTION2 request is placed in Setup[0]. The parameters associated with any particular request are placed in the Parameters vector of the request. The defined subcommand codes are:

Setup[0] Transaction2 Subcommand Code	Value	Meaning
TRANS2_OPEN2	0x00	Create file with extended attributes
TRANS2_FIND_FIRST2	0x01	Begin search for files
TRANS2_FIND_NEXT2	0x02	Resume search for files
TRANS2_QUERY_FS_INFORMATION	0x03	Get file system information
	0x04	Reserved (TRANS_SET_FS_INFORMATION?)
TRANS2_QUERY_PATH_INFORMATION	0x05	Get information about a named file or directory
TRANS2_SET_PATH_INFORMATION	0x06	Set information about a named file or directory
TRANS2_QUERY_FILE_INFORMATION	0x07	Get information about a handle
TRANS2_SET_FILE_INFORMATION	0x08	Set information by handle
TRANS2_FSCTL	0x09	Not implemented by NT server
TRANS2_IOCTL2	0x0A	Not implemented by NT server
TRANS2_FIND_NOTIFY_FIRST	0x0B	Not implemented by NT server
TRANS2_FIND_NOTIFY_NEXT	0x0C	Not implemented by NT server
TRANS2_CREATE_DIRECTORY	Ox0D	Create directory with extended attributes
TRANS2_SESSION_SETUP	0x0E	Session setup with extended security information

Setup[0] Transaction2 Subcommand Code	Value	Meaning
TRANS2_GET_DFS_REFERRAL	0x10	Get a DFS referral
TRANS2_REPORT_DFS_INCONSISTENCY	0x11	Report a DFS knowledge inconsistency

5.3. SMB_COM_NT_TRANSACTION Subcommand Codes

For these transactions, Function in the primary client request indicates the operation to be performed. It may assume one of the following values:

Transaction Subcommand Code	Value	Meaning
NT_TRANSACT_CREATE	1	File open/create
NT_TRANSACT_IOCTL	2	Device IOCTL
NT_TRANSACT_SET_SECURITY_DESC	3	Set security descriptor
NT_TRANSACT_NOTIFY_CHANGE	4	Start directory watch
NT_TRANSACT_RENAME	5	Reserved (Handle-based rename)
NT_TRANSACT_QUERY_SECURITY_DESC	6	Retrieve security descriptor info

5.4. SMB Protocol Dialect Constants

This is the list of CIFS protocol dialects, ordered from least functional (earliest) version to most functional (most recent) version:

Dialect Name	Comment
PC NETWORK PROGRAM 1.0	The original MSNET SMB protocol (otherwise known as the "core protocol")
PCLAN1.0	Some versions of the original MSNET defined this as an alternate to the core protocol name
MICROSOFT NETWORKS 1.03	This is used for the MS-NET 1.03 product. It defines Lock&Read,Write&Unlock, and a special version of raw read and raw write.
MICROSOFT NETWORKS 3.0	This is the DOS LANMAN 1.0 specific protocol. It is equivalent to the LANMAN 1.0 protocol, except the server is required to map errors from the OS/2 error to an appropriate DOS error.
LANMAN1.0	This is the first version of the full LANMAN 1.0 protocol
Windows for Workgroups 3.1a	Windows for Workgroups Version 1.0 (similar to LANMAN1.0 dialect)
LM1.2X002	This is the first version of the full LANMAN 2.0 protocol
DOS LM1.2X002	This is the DOS equivalent of the LM1.2X002 protocol. It is identical to the LM1.2X002 protocol, but the server will perform error mapping to appropriate DOS errors. See section 6.0
DOS LANMAN2.1	DOS LANMAN2.1
LANMAN2.1	OS/2 LANMAN2.1
NT LM 0.12	The SMB protocol designed for NT networking. This has special SMBs which duplicate the NT semantics.

CIFS servers select the most recent version of the protocol known to both client and server. Any CIFS server, which supports dialects newer than the original core dialect, must support all the

messages and semantics of the dialects between the core dialect and the newer one. This is to say that a server, which supports the NT LM 0.12 dialect, must also support all of the messages of the previous 10 dialects. It is the client's responsibility to ensure it only sends SMBs, which are appropriate to the dialect negotiated. Clients must be prepared to receive an SMB response from an earlier protocol dialect -- even if the client used the most recent form of the request.

6. Error Codes and Classes

This section lists all of the valid values for Status.DosError.ErrorClass, and most of the error codes for Status.DosError.Error. Additionally, a mapping between STATUS codes and DOS errors are provided.

The following error classes may be returned by the server to the client.

```
Class       Code      Comment
=======     =====     ========
SUCCESS     0         The request was successful.
ERRDOS      0x01      Error is from the core DOS operating system set.
ERRSRV      0x02      Error is generated by the server network file
                       manager.
ERRHRD      0x03      Error is a hardware error.
ERRCMD      0xFF      Command was not in the "SMB" format.
```

The following error codes may be generated with the SUCCESS error class.

```
Class             Code   Comment
=======           =====  ========
SUCCESS           0      The request was successful.
```

The following error codes may be generated with the ERRDOS error class.

```
Error             Code   Description
======            =====  ============
ERRbadfunc        1      Invalid function.  The server did not
                          recognize or could not perform a system call
                          generated by the server, e.g., set the
                          DIRECTORY attribute on a data file, invalid
                          seek mode.
ERRbadfile        2      File not found.  The last component of a
                          file's pathname could not be found.
ERRbadpath        3      Directory invalid.  A directory component in
                          a pathname could not be found.
ERRnofids         4      Too many open files.  The server has no file
                          handles available.
ERRnoaccess       5      Access denied, the client's context does not
                          permit the requested function.  This includes
                          the following conditions: invalid rename command,
                          write to Fid open for read only, read on Fid
                             open for write only, attempt to delete a
                          non-empty directory
ERRbadfid         6      Invalid file handle.  The file handle
                          specified was not recognized by the server.
ERRbadmcb         7      Memory control blocks destroyed.
ERRnomem          8      Insufficient server memory to perform the
                          requested function.
ERRbadmem         9      Invalid memory block address.
ERRbadenv         10     Invalid environment.
ERRbadformat      11     Invalid format.
ERRbadaccess      12     Invalid open mode.
ERRbaddata        13     Invalid data (generated only by IOCTL calls
```

```
                          within the server).
   ERRbaddrive     15     Invalid drive specified.
   ERRremcd        16     A Delete Directory request attempted to
                           remove the server's current directory.
   ERRdiffdevice   17     Not same device (e.g., a cross volume rename
                           was attempted)
   ERRnofiles      18     A File Search command can find no more files
                           matching the specified criteria.
   ERRbadshare     32     The sharing mode specified for an Open
                           conflicts with existing FIDs on the file.
   ERRlock         33     A Lock request conflicted with an existing
                           lock or specified an invalid mode, or an
                           Unlock requested attempted to remove a lock
                           held by another process.
   ERRfilexists    80     The file named in the request already exists.
   ErrQuota       512    The operation would cause a quota limit to be
                                    exceeded.
   ErrNotALink    513    A link operation was performed on a pathname
                                    that was not a link.
```

The following error codes may be generated with the ERRSRV error class.

```
Error              Code   Description
======             =====  ============
ERRerror           1      Non-specific error code.  It is returned under
                            the following conditions: resource other than
                            disk space exhausted (e.g. TIDs), first SMB
                            command was not negotiate, multiple negotiates
                            attempted, and internal server error.
ERRbadpw           2      Bad password - name/password pair in a Tree
                            Connect or Session Setup are invalid.
ERRaccess          4      The client does not have the necessary access
                            rights within the specified context for the
                            requested function.
ERRinvtid          5      The Tid specified in a command was invalid.
ERRinvnetname      6      Invalid network name in tree connect.
ERRinvdevice       7      Invalid device - printer request made to
                            non-printer connection or non-printer request
                            made to printer connection.
ERRqfull           49     Print queue full (files) -- returned by open
                            print file.
ERRqtoobig         50     Print queue full -- no space.
ERRqeof            51     EOF on print queue dump.
ERRinvpfid         52     Invalid print file FID.
ERRsmbcmd          64     The server did not recognize the command received.
ERRsrverror        65     The server encountered an internal error,
                            e.g., system file unavailable.
ERRbadBID          66     (obsolete)
ERRfilespecs       67     The Fid and pathname parameters contained an
                            invalid combination of values.
ERRbadLink         68     (obsolete)
ERRbadpermits      69     The access permissions specified for a file
                            or directory are not a valid combination.
                            The server cannot set the requested attribute.
ERRbadPID          70
```

```
ERRsetattrmode      71      The attribute mode in the Set File Attribute
                              request is invalid.
ERRpaused           81      Server is paused. (Reserved for messaging)
ERRmsgoff           82      Not receiving messages. (Reserved for messaging)
ERRnoroom           83      No room to buffer message.(Reserved for messaging)
ERRrmuns            87      Too many remote user names.(Reserved for messaging)
ERRtimeout          88      Operation timed out.
ERRnoresource       89      No resources currently available for request.
ERRtoomanyuids      90      Too many Uids active on this session.
ERRbaduid           91      The Uid is not known as a valid user
                              identifier on this session.
ERRusempx           250     Temporarily unable to support Raw, use MPX mode.
ERRusestd           251     Temporarily unable to support Raw,
                              use standard read/write.
ERRcontmpx          252     Continue in MPX mode.
ERRbadPassword      254     (obsolete)
ERR_NOTIFY_ENUM_DIR 1024    Too many files have changed since the last time a
                              NT_TRANSACT_NOTIFY_CHANGE was issued
ERRaccountExpired   2239
ERRbadClient        2240    Cannot access the server from this workstation.
ERRbadLogonTime     2241    Cannot access the server at this time.
ERRpasswordExpired  2242
ERRnosupport        65535   Function not supported.
```

The following error codes may be generated with the ERRHRD error class.

```
Error           Code    Description
======          =====   ============
ERRnowrite      19      Attempt to write on write-protected media
ERRbadunit      20      Unknown unit.
ERRnotready     21      Drive not ready.
ERRbadcmd       22      Unknown command.
ERRdata         23      Data error (CRC).
ERRbadreq       24      Bad request structure length.
ERRseek         25      Seek error.
ERRbadmedia     26      Unknown media type.
ERRbadsector    27      Sector not found.
ERRnopaper      28      Printer out of paper.
ERRwrite        29      Write fault.
ERRread         30      Read fault.
ERRgeneral      31      General failure.
ERRbadshare     32      A open conflicts with an existing open.
ERRlock         33      A Lock request conflicted with an existing
                         lock or specified an invalid mode, or an
                         Unlock requested attempted to remove a lock
                         held by another process.
ERRwrongdisk    34      The wrong disk was found in a drive.
ERRFCBUnavail   35      No FCBs are available to process request.
ERRsharebufexc  36      A sharing buffer has been exceeded.
```

These are the mappings of the listed STATUS_codes to the DOS errors.

DOS Error	Status Code
ERROR_INVALID_FUNCTION	STATUS_NOT_IMPLEMENTED
ERROR_FILE_NOT_FOUND	STATUS_NO_SUCH FILE_
ERROR_PATH_NOT_FOUND	STATUS_OBJECT_PATH_NOT_FOUND
ERROR_TOO_MANY_OPEN_FILES	STATUS_TOO_MANY_OPENED_FILES
ERROR_ACCESS_DENIED	STATUS_ACCESS_DENIED
ERROR_INVALID_HANDLE	STATUS_INVALID_HANDLE
ERROR_NOT_ENOUGH_MEMORY	STATUS_INSUFFICIENT_RESOURCES
ERROR_INVALID_ACCESS	STATUS_ACCESS_DENIED
ERROR_INVALID_DATA	STATUS_DATA_ERROR
ERROR_CURRENT_DIRECTORY	STATUS_DIRECTORY_NOT_EMPTY
ERROR_NOT_SAME_DEVICE	STATUS_NOT_SAME_DEVICE
ERROR_NO_MORE_FILES	STATUS_NO_MORE_FILES
ERROR_WRITE_PROTECT	STATUS_MEDIA_WRITE_PROTECTED
ERROR_NOT_READY	STATUS_DEVICE_NOT_READY
ERROR_CRC	STATUS_CRC_ERROR
ERROR_BAD_LENGTH	STATUS_DATA_ERROR
ERROR_NOT_DOS_DISK	STATUS_DISK_CORRUPT_ERROR
ERROR_SECTOR_NOT_FOUND	STATUS_NONEXISTENT_SECTOR
ERROR_OUT_OF_PAPER	STATUS_DEVICE_PAPER_EMPTY
ERROR_SHARING_VIOLATION	STATUS_SHARING_VIOLATION
ERROR_LOCK_VIOLATION	STATUS_FILE_LOCK_CONFLICT
ERROR_WRONG_DISK	STATUS_WRONG_VOLUME
ERROR_NOT_SUPPORTED	STATUS_NOT_SUPPORTED
ERROR_REM_NOT_LIST	STATUS_REMOTE_NOT_LISTENING
ERROR_DUP_NAME	STATUS_DUPLICATE_NAME
ERROR_BAD_NETPATH	STATUS_BAD_NETWORK_PATH
ERROR_NETWORK_BUSY	STATUS_NETWORK_BUSY
ERROR_DEV_NOT_EXIST	STATUS_DEVICE_DOES_NOT_EXIST
ERROR_TOO_MANY_CMDS	STATUS_TOO_MANY_COMMANDS
ERROR_ADAP_HDW_ERR	STATUS_ADAPTER_HARDWARE_ERROR
ERROR_BAD_NET_RESP	STATUS_INVALID_NETWORK_RESPONSE
ERROR_UNEXP_NET_ERR	STATUS_UNEXPECTED_NETWORK_ERROR
ERROR_BAD_REM_ADAP	STATUS_BAD_REMOTE_ADAPTER
ERROR_PRINTQ_FULL	STATUS_PRINT_QUEUE_FULL
ERROR_NO_SPOOL_SPACE	STATUS_NO_SPOOL_SPACE
ERROR_PRINT_CANCELLED	STATUS_PRINT_CANCELLED
ERROR_NETNAME_DELETED	STATUS_NETWORK_NAME_DELETED
ERROR_NETWORK_ACCESS_DENIED	STATUS_NETWORK_ACCESS_DENIED
ERROR_BAD_DEV_TYPE	STATUS_BAD_DEVICE_TYPE
ERROR_BAD_NET_NAME	STATUS_BAD_NETWORK_NAME
ERROR_TOO_MANY_NAMES	STATUS_TOO_MANY_NAMES
ERROR_TOO_MANY_SESS	STATUS_TOO_MANY_SESSIONS
ERROR_SHARING_PAUSED	STATUS_SHARING_PAUSED
ERROR_REQ_NOT_ACCEP	STATUS_REQUEST_NOT_ACCEPTED
ERROR_REDIR_PAUSED	STATUS_REDIRECTOR_PAUSED
ERROR_FILE_EXISTS	STATUS_OBJECT_NAME_COLLISION
ERROR_INVALID_PASSWORD	STATUS_WRONG_PASSWORD
ERROR_INVALID_PARAMETER	STATUS_INVALID_PARAMETER
ERROR_NET_WRITE_FAULT	STATUS_NET_WRITE_FAULT
ERROR_BROKEN_PIPE	STATUS_PIPE_BROKEN
ERROR_OPEN_FAILED	STATUS_OPEN_FAILED
ERROR_BUFFER_OVERFLOW	STATUS_BUFFER_OVERFLOW
ERROR_DISK_FULL	STATUS_DISK_FULL
ERROR_SEM_TIMEOUT	STATUS_IO_TIMEOUT
ERROR_INSUFFICIENT_BUFFER	STATUS_BUFFER_TOO_SMALL
ERROR_INVALID_NAME	STATUS_OBJECT_NAME_INVALID

ERROR_INVALID_LEVEL	STATUS_INVALID_LEVEL
ERROR_BAD_PATHNAME	STATUS_OBJECT_PATH_INVALID
ERROR_BAD_PIPE	STATUS_INVALID_PARAMETER
ERROR_PIPE_BUSY	STATUS_PIPE_NOT_AVAILABLE
ERROR_NO_DATA	STATUS_PIPE_EMPTY
ERROR_PIPE_NOT_CONNECTED	STATUS_PIPE_DISCONNECTED
ERROR_MORE_DATA	STATUS_BUFFER_OVERFLOW
ERROR_VC_DISCONNECTED	STATUS_VIRTUAL_CIRCUIT_CLOSED
ERROR_INVALID_EA_NAME	STATUS_INVALID_EA_NAME
ERROR_EA_LIST_INCONSISTENT	STATUS_EA_LIST_INCONSISTENT
ERROR_EAS_DIDNT_FIT	STATUS_EA_TOO_LARGE
ERROR_EA_FILE_CORRUPT	STATUS_EA_CORRUPT_ERROR
ERROR_EA_TABLE_FULL	STATUS_EA_CORRUPT_ERROR
ERROR_INVALID_EA_HANDLE	STATUS_EA_CORRUPT_ERROR

7. Security Considerations

MISSING

Suggested content for this section:

Define share security level. What dialects support it?

1. Define user security level.
2. How is it supported in PDC/BDC environment (NT4)
3. How it supported in Active directory environment. Define the different security considerations in different Active Directory modes.
4. How Kerberos security is used?
5. What are the protocols (or DCE/RPC) needed for each of the User level security models
6. Some discussion on how file access is authenticated, or how the SID is retrieved in each of the user level environments mentioned above for ACL
7. Include the security protocol, or reference to it

8. References

[1] P. Mockapetris, "Domain Names - Concepts And Facilities", RFC 1034, November 1987

[2] P. Mockapetris, "Domain Names - Implementation And Specification", RFC 1035, November 1987

[3] Karl Auerbach, "Protocol Standard For A Netbios Service On A TCP/UDP Transport: Concepts And Methods", RFC 1001, March 1987

[4] Karl Auerbach, "Protocol Standard For A Netbios Service On A TCP/UDP Transport: Detailed Specifications", RFC 1002, March 1987

[5] US National Bureau of Standards, "Data Encryption Standard", Federal Information Processing Standard (FIPS) Publication 46-1, January 1988

[6] Rivest, R. - MIT and RSA Data Security, Inc., "The MD4 Message Digest Algorithm", RFC 1320, April 1992

[7] Rivest, R. – MIT and RSA Data Security, Inc., “The MD5 Message-Digest Algorithm”, RFC 1321, April 1992

[8] Metzger, P. Piermont, Simpson, W. Daydreamer, “IP Authentication using Keyed MD5”, RFC 1828, August 1995

[9] Leach, P. – Microsoft, “CIFS Authentication Protocols Specification, Author’s Draft 4

[10] B. Kaliski, M.Robshaw, "Message Authentication with MD5", CryptoBytes, Spring 1995, RSA Inc, (ftp://ftp.rsasecurity.com/pub/cryptobytes/crypto1n1.pdf)

[11] X/Open Company Ltd., "X/Open CAE Specification - Protocols for X/Open PC Interworking: SMB, Version 2", X/Open Document Number: CAE 209, September 1992.

9. Appendix A -- NETBIOS transport over TCP

With respect to the 7-layer OSI reference model, NetBIOS is a session layer (layer 5) Application Programmer's Interface (API). The NetBIOS API has been implemented on top of a variety of transports (layer 4), including TCP/IP. NetBIOS over TCP/IP transport is specified in RFC 1001 and RFC 1002 (IETF Standard #19).

NetBIOS is the traditional session layer interface for SMB/CIFS. For backward compatibility with older systems, CIFS implementations SHOULD provide support for RFC 1001/1002 transport.

9.1. Connection Establishment

Connections are established and messages transferred via the NetBIOS session service (see section 5.3 of RFC 1001 and section 4.3 of RFC 1002). The system that originates the connection is the "calling" node; the target node is the "called" node. In order to establish an SMB session, a TCP connection must be established between the calling and called nodes. If a TCP connection already exists, the SMB session may make use of the existing connection.

9.2. Connecting to a server using the NetBIOS name

Before a NetBIOS session can be established, the node initiating the session (the "calling" node) must discover the IP address of the target node (the "called" node). This is done using the NetBIOS name service (see section 5.2 of RFC 1001 and section 4.2 of RFC 1002). NetBIOS names are always 16 bytes, padded with spaces (0x20) if necessary, as specified in the RFCs. The 16th byte has been reserved, however, for use as a service indicator. This field is known as the "suffix byte".

The NetBIOS session service requires that the client provide the NetBIOS names of both the calling and called nodes. The calling name is the default NetBIOS name of the client, space padded as described, with a suffix byte value of 0x00. The called name is the NetBIOS name of the server with a suffix byte value of 0x20. Server implementations which support SMB via NetBIOS over TCP/IP MUST support the registration and use of the server NetBIOS name.

The calling name is not significant in CIFS, except that an identical name from the same transport address is assumed to represent the same client. SMB session establishment is initiated using a "Session Request" packet sent to port 139 (see section 4.3.2 of RFC 1002).

9.3. Connecting to a server using a DNS name or IP address

Implementations MAY support the use of DNS names or IP addresses in addition to NetBIOS names when initiating SMB connections via NetBIOS over TCP/IP transport. This functionality is an extension to the NetBIOS over TCP/IP behavior specified in RFC 1001 and RFC 1002, and is not part of that standard.

As stated above, the Session Request packet requires a called and a calling name, both of which are NetBIOS names. In order to create a Session Request packet, the DNS name or IP address of the server must be reverse-mapped to the server's NetBIOS name. Mechanisms for doing so are as follows:

9.3.1. NetBIOS Adapter Status

A NetBIOS Adapter Status Query is sent to the target IP address. If a response is received and the target is offering SMB services via NetBIOS over TCP, then the response will include a NetBIOS name with a suffix byte value of 0x20. This NetBIOS name may be used as the called name in a Session Request packet.

9.3.2. Generic Server Name

Servers offering SMB services via NetBIOS over TCP/IP MAY accept the generic SMB server name "*SMBSERVER". A client can simply use the name "*SMBSERVER" as the called name in a Session Request packet. As with all SMB server NetBIOS names, the "*SMBSERVER" name must be space padded and terminated with a suffix byte value of 0x20.

The "*SMBSERVER" name MUST NOT be registered with the NetBIOS name service, as it is an illegal NetBIOS name (see section 5.2 of RFC 1001).

The target may return a CALLED NAME NOT PRESENT error. This may simply indicate that the server does not support the "*SMBSERVER" generic name.

9.3.3. - Parsing the DNS Name (guessing)

Systems which support NetBIOS transport over TCP/IP will often use the same base name within the DNS and NetBIOS name spaces. Thus, the first label of the DNS name represents a good guess at the NetBIOS name of the server.

The first label of the DNS name consists of the initial portion of the DNS name string, up to but not including the first dot character ('.'). If the label is greater than 15 bytes in length, it must be truncated to 15 bytes. The result is then space padded to a total of 15 bytes, and a suffix value 0x20 is used. This forms a valid NetBIOS name that may be used as a called name in a Session Request packet.

If the target returns a CALLED NAME NOT PRESENT error, then the DNS name guess is incorrect. If the original user input was an IP address, the DNS name can be determined using a reverse lookup against the DNS. Any or all of the above MAY be tried in any order.

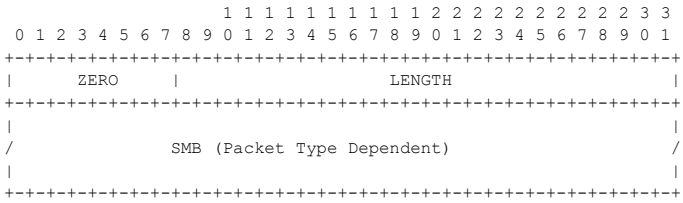
9.4. NetBIOS Name character set

There is no standard character set for NetBIOS names. NetBIOS names are simply strings of octets, with the following restrictions:

- Names which are to be registered with the NetBIOS Name Service must not begin with an asterisk (0x2A). (The *SMBSERVER name is never registered.)
- Names should not contain a NUL (0x00) octet. Common implementation languages may interpret the NUL octet value as a string terminator.

10. Appendix B -- TCP transport

When operating CIFS over TCP, connections are established to TCP port 445, and each message is framed as follows:



Each CIFS request starts with a 4 byte field encoded as above: a byte of zero, followed by three bytes of length; after that follows the body of the request.

11. Appendix C – Share Level Server Security

Each server makes a set of resources available to clients on the network. A resource being shared may be a directory tree, named pipe, printer, etc. As far as clients are concerned, the server has no storage or service dependencies on any other servers; a client considers the server to be the sole provider of the file (or other resource) being accessed.

The CIFS protocol requires server authentication of users before file accesses are allowed, and each server authenticates its own users. A client system must send authentication information to the server before the server will allow access to its resources.

The CIFS protocol used to define two methods that can be selected by the server for security: *share level* and *user level*. User level security is the only non-obsolescent method.

A *share level* server makes some directory on a disk device (or other resource) available. An optional password may be required to gain access. Thus, any user on the network who knows the name of the server, the name of the resource, and the password has access to the resource. Share level security servers may use different passwords for the same shared resource with different passwords, allowing different levels of access.

Share-level-only clients do not send SESSION_SETUP_ANDX requests. Instead, they send TREE_CONNECT_ANDX requests that include a password or use challenge/response authentication to prove that they know a password.

When a *user level* server validates the account name and password presented by the client, an identifier representing that authenticated instance of the user is returned to the client in the *Uid* field of the response SMB. In contrast, a *share level* server returns no useful information in the *Uid* field.

If the server is executing in share level security mode, *Tid* is the only thing used to allow access to the shared resource. Thus, if the user is able to perform a successful connection to the server specifying the appropriate netname and passwd (if any), the resource may be accessed according to the access rights associated with the shared resource (same for all who gained access this way).

The user level security model was added after the original dialect of the CIFS protocol was issued, and subsequently some clients may not be capable of sending account name and passwords to the server. A server in user level security mode communicating with one of these clients *may* allow a client to connect to resources even if the client has not sent account name information:

1) If the client's computer name is identical to an account name known on the server, and if the password supplied or authenticated via challenge/response to connect to the shared resource matches that account's password, an implicit "user logon" will be performed using those values. If the above fails, the server may fail the request or assign a default account name of its choice.
2) The value of *Uid* in subsequent requests by the client will be ignored, and all access will be validated assuming the account name selected above.

12. Appendix D – CIFS UNIX Extension

12.1. Introduction

The purpose of these extensions is to allow UNIX based CIFS clients and servers to exchange information used by UNIX systems, but not present in Windows based CIFS servers or clients. These extensions may not be implemented by all UNIX systems. Two simple examples are symbolic links and UNIX special files (e.g. UNIX named pipes).

The CIFS UNIX Extension are intended for use by all UNIX and UNIX-like systems the implement the CIFS protocol.

12.2. Principles

These are a set of principles that the extensions meet.

Minimal changes	To make the extensions easier to implement, the number of changes and additions were minimized.
Can be implemented on non-UNIX systems	While being useful for UNIX, the extension allow one end of the connection to be a non-UNIX system. This is so that other CIFS servers and clients can better integrate with a UNIX CIFS client or server.
Use current commands	The changes only affect current commands. There was no need for UNIX CIFS clients to use CIFS commands marked as obsolete, nor should there be any changes to obsolete requests.
Retain existing CIFS semantics	The existing semantics of CIFS are retained. Perhaps the most notable is that file names are case insensitive, but case should be preserved.
Use CIFS security model	The standard CIFS security model is still used. This requires each distinct user to be logged into the server.
Addition to dialect	This specification is an addition to the CIFS dialect, currently NT LM 0.12. It is selected by the capability bit in the server's Negotiate protocol response.
Future resilient	Future enhancements MUST not modify or change the meaning of previous implementations of the specification.

12.3. CIFS Protocol Modifications

This section details the require changes to the CIFS protocol that are needed to support CIFS UNIX Extensions. A summary of the changes is listed below.

In the Negotiate Protocol SMB reserve a capabilities bit, CAP_UNIX with the value of 0x00800000, in the Server capabilities field to indicate support of CIFS Extension for UNIX.

Reserve information levels numbers 0x200-0x2FF

TRANS2_QUERY_FS_INFORMATION, TRANS2_QUERY_PATH_INFO, TRANS2_QUERY_FILE_INFO, TRANS2_SET_PATH_INFO, TRANS2_SET_FILE_INFO, TRANS2_FINDFIRST, and TRANS2_FINDNEXT SMBs for CIFS Extensions for UNIX.

12.4. Modified SMBs

SMB	Modification
NEGOTIATE	Added CAP_UNIX (0x00800000) to the server capabilities field. See 4.2
TRANS2_QUERY_FS_INFORMATION	Added Following Information Levels: SMB_QUERY_CIFS_UNIX_INFO (0x200) See 4.1.6.7
TRANS2_QUERY_PATH_INFORMATION	Added Following Information Levels: SMB_QUERY_FILE_UNIX_BASIC (0x200) See 4.2.15.12 SMB_QUERY_FILE_UNIX_LINK (0x201) See 4.2.15.13
TRANS2_QUERY_FILE_INFORMATION	Same modification as done in TRANS2_QUERY_PATH_INFORMATION
TRANS2_SET_PATH_INFORMATION	Added Following Information Levels: SMB_SET_FILE_UNIX_BASIC (0x200) See 4.2.17.3 SMB_SET_FILE_UNIX_LINK (0x201) See 4.2.17.4 SMB_SET_FILE_UNIX_HLINK (0X203) See 4.2.17.5
TRANS2_SET_FILE_INFORMATION	Same modification as done in TRANS2_SET_PATH_INFORMATION
TRANS2_FINDFIRST	Added following Information Levels: SMB_FIND_FILE_UNIX (0X202) See 4.3.4.8
TRANS2_FINDNEXT	Same modification as done in TRANS2_FINDFIRST

12.5. Guidelines for implementers

- Once the Client determines that the server supports the CIFS UNIX Extension it should first send SMB_QUERY_CIFS_UNIX_INFO before sending any other CIFS UNIX Extension SMBs to determine the version and capabilities that are supported by the server.
- Clients or servers using this extension should have no specific reserved filenames (e.g. CON, AUX, PRN), and should not need to take specific action to protect the other end of the connection from them. If they have any such requirements, they must do them internally. This also applies to reserved characters in filenames (e.g. : \ |).
- Inodes can be transferred in the uniqueid field of SMB_QUERY_FILE_UNIX_BASIC (0x200).
- Clients should operate in UNICODE if at all possible. A useful bridging step is to implement UTF-8
- Symbolic links are created by calling TRANS2_SET_PATH_INFO with the SMB_QUERY_FILE_UNIX_LINK infolevel data structure provided.
- Device file (and other special UNIX files) are created by calling TRANS2_SET_PATH_INFO with the SMB_QUERY_FILE_UNIX_BASIC infolevel data structure appropriately filled in for a device node.

- Servers should return their timezone as UTC. This will then require no timezone mapping by the client or server. The NetRemoteTimeOfDay IPC should still return the real local time.
- Creates with particular permissions can be achieved by sending a CREATE_AND_X and a TRANS2_SET_PATH_INFO SMBs.

13. Appendix E – CIFS Macintosh Extension

13.1. Introduction

The purpose of these extensions is to allow the Macintosh to better interoperate in a CIFS network. With these extensions Macintosh Clients will be able to reduce network traffic generated by the Macintosh, which in turn would speed up file access by the Client. These extensions will allow non-Macintosh Clients access to Macintosh files and also allow for the server to decide how to store Macintosh files and folders.

The CIFS Macintosh Extension is intended for use by all systems that implement the CIFS protocol.

13.2. Principles

These are a set of principles that the extensions meet.

Minimal changes	To make the extensions easier to implement, the number of changes and additions were minimized.
Can be implemented on non-Macintosh systems	While being useful for Macintosh, the extension allows one end of the connection to be a non-Macintosh system. This is so that other CIFS servers and clients can better integrate with a Macintosh CIFS client or server.
Use current commands	The changes only affect current commands. There is no need for CIFS clients to use CIFS commands marked as obsolete, nor should there be any changes to obsolete requests.
Retain existing CIFS semantics	The existing semantics of CIFS are retained.
Use CIFS security model	The standard CIFS security model is still used. This requires each distinct user to be logged into the server.
Addition to dialect	These items are an addition to the CIFS dialect, currently NT LM 0.12. These extensions are turn on by the server responding with out an error to the TRANS2_QUERY_FS_INFORMATION call with an info level of Trans2_GetSMB_MAC_QUERY_FS_INFO.
Future resilient	Future enhancements MUST not modify or change the meaning of previous implementations of the specification.

13.3. CIFS Protocol Modifications

This section details the require changes to the CIFS protocol that are needed to support CIFS Macintosh Extensions. These extensions require support of the NT LM 0.12 dialect with some minor additions. The Server must support the NT stream format for the opening of the resource, comments, and data streams of a file. A summary of the changes is listed below.

Reserve information levels numbers 0x300-0x3FF in the TRANS2_QUERY_FS_INFORMATION, TRANS2_QUERY_PATH_INFO, TRANS2_SET_PATH_INFO, TRANS2_FINDFIRST, and TRANS2_FINDNEXT SMBs for CIFS Extensions for Macintosh.

13.4. Modified SMBs

SMB	Modification
TRANS2_QUERY_FS_INFORMATION	Added Following Information Levels: SMB_ QUERY_ MAC_FS_INFO (0x301) See 4.1.6.7
TRANS2_FINDFIRST	Added following Information Levels: SMB_ FINDBOTH_ MAC_HFS_INFO (0X302) See 4.3.4.9
TRANS2_FINDNEXT	Same modification as done in TRANS2_FINDFIRST
TRANS2_SET_PATH_INFORMATION	Added Following Information Levels: SMB_MAC_SET_FINDER_INFO (0x303) See 4.2.18.6 SMB_MAC_DT_ADD_APPL (0x304) See 4.2.18.7 SMB_MAC_DT_REMOVE_APPL (0x305) See 4.2.18.8 SMB_MAC_DT_ADD_ICON (0x309) See 4.2.18.9
TRANS2_QUERY_PATH_INFORMATION	Added Following Information Levels: SMB_MAC_DT_GET_APPL (0x306) See 4.2.16.14 SMB_MAC_DT_GET_ICON (0x307) See 4.2.16.15 SMB_MAC_DT_GET_ICON_INFO (0x308) See 4.2.16.16

13.5. Guidelines for implementers

- These extensions will be processed on share-by-share bases. This means that the Client will have to confirm that each share supports these extensions not just that the Server supports these extensions. This will allow a server to have some shares that are Macintosh aware and others that are not.
- When a file or folder is deleted then all streams and information stored on the sever associated with that file or folder should be removed. When a file or folder is Copied/Renamed/Moved then all streams and information stored on the sever associated with that file or folder should be Copied/Renamed/Moved.
- Clients or servers using this extension should have no specific reserved filenames (e.g. CON, AUX, PRN), and should not need to take specific action to protect the other end of the connection from them. If they have any such requirements, they must do them internally. This also applies to reserved characters in filenames (e.g. : \ |).
- Clients should operate in UNICODE if at all possible.
- Supporting the Desktop Database calls requires having a way to store information in a database. There are two kinds of information store in the database. Applications path that is associated with an application signature. Icons are stored based on size, icon type, file creator, and file type.

14. Appendix F – API Numbers for Transact based RAP calls

API_WshareEnum	0
API_WshareGetInfo	1
API_WshareSetInfo	2
API_WshareAdd	3
API_WshareDel	4
API_NetShareCheck	5
API_WsessionEnum	6
API_WsessionGetInfo	7
API_WsessionDel	8
API_WconnectionEnum	9
API_WfileEnum	10
API_WfileGetInfo	11
API_WfileClose	12
API_WserverGetInfo	13
API_WserverSetInfo	14
API_WserverDiskEnum	15
API_WserverAdminCommand	16
API_NetAuditOpen	17
API_WauditClear	18
API_NetErrorLogOpen	19
API_WerrorLogClear	20
API_NetCharDevEnum	21
API_NetCharDevGetInfo	22
API_WCharDevControl	23
API_NetCharDevQEnum	24
API_NetCharDevQGetInfo	25
API_WCharDevQSetInfo	26
API_WCharDevQPurge	27
API_WCharDevQPurgeSelf	28
API_WMessageNameEnum	29
API_WMessageNameGetInfo	30
API_WMessageNameAdd	31
API_WMessageNameDel	32
API_WMessageNameFwd	33
API_WMessageNameUnFwd	34
API_WMessageBufferSend	35
API_WMessageFileSend	36
API_WMessageLogFileSet	37
API_WMessageLogFileGet	38
API_WServiceEnum	39
API_WServiceInstall	40
API_WServiceControl	41
API_WAccessEnum	42
API_WAccessGetInfo	43
API_WAccessSetInfo	44
API_WAccessAdd	45
API_WAccessDel	46
API_WGroupEnum	47
API_WGroupAdd	48
API_WGroupDel	49
API_WGroupAddUser	50
API_WGroupDelUser	51
API_WGroupGetUsers	52
API_WUserEnum	53
API_WUserAdd	54
API_WUserDel	55
API_WUserGetInfo	56
API_WUserSetInfo	57

API_WUserPasswordSet 58
API_WUserGetGroups 59
API_DeadTableEntry 60
/*This line and number replaced a Dead Entry */
API_WWkstaSetUID 62
API_WWkstaGetInfo 63
API_WWkstaSetInfo 64
API_WUseEnum 65
API_WUseAdd 66
API_WUseDel 67
API_WUseGetInfo 68
API_WPrintQEnum 69
API_WPrintQGetInfo 70
API_WPrintQSetInfo 71
API_WPrintQAdd 72
API_WPrintQDel 73
API_WPrintQPause 74
API_WPrintQContinue 75
API_WPrintJobEnum 76
API_WPrintJobGetInfo 77
API_WPrintJobSetInfo_OLD 78
/* This line and number replaced a Dead Entry */
/* This line and number replaced a Dead Entry */
API_WPrintJobDel 81
API_WPrintJobPause 82
API_WPrintJobContinue 83
API_WPrintDestEnum 84
API_WPrintDestGetInfo 85
API_WPrintDestControl 86
API_WProfileSave 87
API_WProfileLoad 88
API_WStatisticsGet 89
API_WStatisticsClear 90
API_NetRemoteTOD 91
API_WNetBiosEnum 92
API_WNetBiosGetInfo 93
API_NetServerEnum 94
API_I_NetServerEnum 95
API_WServiceGetInfo 96
/* This line and number replaced a Dead Entry */
/* This line and number replaced a Dead Entry */
/* This line and number replaced a Dead Entry */
/* This line and number replaced a Dead Entry */
/* This line and number replaced a Dead Entry */
/* This line and number replaced a Dead Entry */
API_WPrintQPurge 103
API_NetServerEnum2 104
API_WAccessGetUserPerms 105
API_WGroupGetInfo 106
API_WGroupSetInfo 107
API_WGroupSetUsers 108
API_WUserSetGroups 109
API_WUserModalsGet 110
API_WUserModalsSet 111
API_WFileEnum2 112
API_WUserAdd2 113
API_WUserSetInfo2 114
API_WUserPasswordSet2 115
API_I_NetServerEnum2 116
API_WConfigGet2 117

API_WConfigGetAll2 118
API_WGetDCName 119
API_NetHandleGetInfo 120
API_NetHandleSetInfo 121
API_WStatisticsGet2 122
API_WBuildGetInfo 123
API_WFileGetInfo2 124
API_WFileClose2 125
API_WNetServerReqChallenge 126
API_WNetServerAuthenticate 127
API_WNetServerPasswordSet 128
API_WNetAccountDeltas 129
API_WNetAccountSync 130
API_WUserEnum2 131
API_WWkstaUserLogon 132
API_WWkstaUserLogoff 133
API_WLogonEnum 134
API_WErrorLogRead 135
API_WI_NetPathType 136
API_WI_NetPathCanonicalize 137
API_WI_NetPathCompare 138
API_WI_NetNameValidate 139
API_WI_NetNameCanonicalize 140
API_WI_NetNameCompare 141
API_WAuditRead 142
API_WPrintDestAdd 143
API_WPrintDestSetInfo 144
API_WPrintDestDel 145
API_WUserValidate2 146
API_WPrintJobSetInfo 147
API_TI_NetServerDiskEnum 148
API_TI_NetServerDiskGetInfo 149
API_TI_FTVerifyMirror 150
API_TI_FTAbortVerify 151
API_TI_FTGetInfo 152
API_TI_FTSetInfo 153
API_TI_FTLockDisk 154
API_TI_FTFixError 155
API_TI_FTAbortFix 156
API_TI_FTDiagnoseError 157
API_TI_FTGetDriveStats 158
/* This line and number replaced a Dead Entry */
API_TI_FTErrorGetInfo 160
/* This line and number replaced a Dead Entry */
/* This line and number replaced a Dead Entry */
API_NetAccessCheck 163
API_NetAlertRaise 164
API_NetAlertStart 165
API_NetAlertStop 166
API_NetAuditWrite 167
API_NetIRemoteAPI 168
API_NetServiceStatus 169
API_I_NetServerRegister 170
API_I_NetServerDeregister 171
API_I_NetSessionEntryMake 172
API_I_NetSessionEntryClear 173
API_I_NetSessionEntryGetInfo 174
API_I_NetSessionEntrySetInfo 175
API_I_NetConnectionEntryMake 176
API_I_NetConnectionEntryClear 177

API_I_NetConnectionEntrySetInfo 178
API_I_NetConnectionEntryGetInfo179
API_I_NetFileEntryMake 180
API_I_NetFileEntryClear 181
API_I_NetFileEntrySetInfo 182
API_I_NetFileEntryGetInfo 183
API_AltSrvMessageBufferSend 184
API_AltSrvMessageFileSend 185
API_wI_NetRplWkstaEnum 186
API_wI_NetRplWkstaGetInfo 187
API_wI_NetRplWkstaSetInfo 188
API_wI_NetRplWkstaAdd 189
API_wI_NetRplWkstaDel 190
API_wI_NetRplProfileEnum 191
API_wI_NetRplProfileGetInfo 192
API_wI_NetRplProfileSetInfo 193
API_wI_NetRplProfileAdd 194
API_wI_NetRplProfileDel 195
API_wI_NetRplProfileClone 196
API_wI_NetRplBaseProfileEnum 197
/* This line and number replaced a Dead Entry */
/* This line and number replaced a Dead Entry */
/* This line and number replaced a Dead Entry */
API_WIServerSetInfo 201
/* This line and number replaced a Dead Entry */
/* This line and number replaced a Dead Entry */
/* This line and number replaced a Dead Entry */
API_WPrintDriverEnum 205
API_WPrintQProcessorEnum 206
API_WPrintPortEnum 207
API_WNetWriteUpdateLog 208
API_WNetAccountUpdate 209
API_WNetAccountConfirmUpdate 210
API_WConfigSet 211
API_WAccountsReplicate 212
/* 213 is used by WfW */
API_SamOEMChgPasswordUser2_P 214
API_NetServerEnum3 215
API_WprintDriverGetInfo 250
API_WprintDriverSetInfo 251
API_WaliasAdd 252
API_WaliasDel 253
API_WaliasGetInfo 254
API_WaliasSetInfo 255
API_WaliasEnum 256
API_WuserGetLogonAsn 257
API_WuserSetLogonAsn 258
API_WuserGetAppSel 259
API_WuserSetAppSel 260
API_WappAdd 261
API_WappDel 262
API_WappGetInfo 263
API_WappSetInfo 264
API_WappEnum 265
API_WUserDCDBInit 266
API_WDASDAdd 267
API_WDASDDel 268
API_WDASDGetInfo 269
API_WDASDSetInfo 270
API_WDASDEnum 271

API_WDASDCheck	272
API_WDASDCtl	273
API_WuserRemoteLogonCheck	274
API_WUserPasswordSet3	275
API_WCreateRIPLMachine	276
API_WDeleteRIPLMachine	277
API_WGetRIPLMachineInfo	278
API_WSetRIPLMachineInfo	279
API_WEnumRIPLMachine	280
API_WI_ShareAdd	281
API_WI_AliasEnum	282
API_WaccessApply	283
API_WPrt16Query	284
API_WPrt16Set	285
API_WUserDel100	286
API_WUserRemoteLogonCheck2	287
API_WRemoteTODSet	294
API_WprintJobMoveAll	295
API_W16AppParmAdd	296
API_W16AppParmDel	297
API_W16AppParmGet	298
API_W16AppParmSet	299
API_W16RIPLMachineCreate	300
API_W16RIPLMachineGetInfo	301
API_W16RIPLMachineSetInfo	302
API_W16RIPLMachineEnum	303
API_W16RIPLMachineListParmEnum	304
API_W16RIPLMachClassGetInfo	305
API_W16RIPLMachClassEnum	306
API_W16RIPLMachClassCreate	307
API_W16RIPLMachClassSetInfo	308
API_W16RIPLMachClassDelete	309
API_W16RIPLMachClassLPEnum	310
API_W16RIPLMachineDelete	311
API_W16WSLevelGetInfo	312
API_WserverNameAdd	313
API_WserverNameDel	314
API_WserverNameEnum	315
API_I_WDASDEnum	316
API_I_WDASDEnumTerminate	317
API_I_WDASDSetInfo2	318
MAX_API	318