Network Working Group
INTERNET-DRAFT
draft-leach-cifs-v1-spec-01.txt
Category: Informational
Expires June 19, 1998

Paul J. Leach, Microsoft
Dilip C. Naik, Microsoft

December 19, 1997

A Common Internet File System (CIFS/1.0) Protocol

Preliminary Draft

Status of this Memo

Distribution of this document is unlimited. Please send comments to the authors at <cifs@microsoft.com>. Discussion of CIFS is on the mailing list <CIFS@DISCUSS.MICROSOFT.COM>; subscribe by sending a message to <ListServ@CIFS@DISCUSS.MICROSOFT.COM> with a body of "subscribe CIFS you@your.domain". The mailing list archives are at <http://discuss.microsoft.com/archives/cifs.html. There is a CIFS home page at <http://www.microsoft.com/intdev/cifs>.

Abstract

This document describes the CIFS file sharing protocol, version 1.0. 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 omits discussion of obsolescent requests not needed by modern clients. They are defined in a companion document Obsolescent SMB Requests.

Table Of Contents

1 Introduction

This document describes the file sharing protocol for a proposed 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 specification 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 specification is limited to describing requests and responses for file services. Separate specifications 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:

o File access

o File and record locking

o Safe caching, read-ahead, and write-behind

o File change notification

o Protocol version negotiation

o Extended attributes

o Distributed replicated virtual volumes

o Server name resolution independence

o Batched requests

o 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 can not 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.

2 Protocol Operation Overview

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

- o Parse the full file name to determine the server name, and the relative name within that server.
- o Resolve the server name to a transport address (this may be cached)
- o Make a connection to the server (if no connection is already available)
- o 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 specification. 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 specification. 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 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 there are really only three round trips in the sequence, and the last one can be done asynchronously by the client.

Client Command	Server Response
SMB_COM_NEGOTIATE	Must be the first message sent by 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 the client wants to access. 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 an CIFS client to an 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

- 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 hard 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

Network performance can be increased if a client does not need to inform the server immediately about every change it makes to a file, or have to worry that other clients can make its information about the file out of date. For example, a client does not have to immediately write information into a file on the server if the client knows that no other process is accessing the data. Likewise, the client can buffer read-ahead data from the file if the client knows that no other process is writing the data.

The mechanism which allows clients to dynamically alter their buffering strategy in a consistent manner is knows as "opportunistic locks", or oplocks for short. Versions of the CIFS file sharing protocol including and newer than the "LANMAN1.0" dialect support oplocks. (Note, however, that an implementation, even of these later dialects, can implement oplocks trivially by always refusing to grant them.)

There are three different types of oplocks:

o A Level II oplock, when held, informs a client that there are multiple concurrent clients of a file, and none has yet modified it. It allows the client to perform reads and file attribute fetches using cached or read-ahead local information, but all other requests have to be sent to the server.

o An exclusive oplock, when held, informs a client that it is the only one to have a file open. It allows the client to perform all file operations using cached or read-ahead local information until it closes the file, at which time the server has to be updated with any changes made to the state of the file (contents and attributes).

o A batch oplock, when held, informs a client that it is the only one to have a file open. It allows the client to perform all file operations on cached or read-ahead local information (including opens and closes).

If a client holds no oplocks, all requests other than reads must be sent to the server. Reads may be performed using cached or read-ahead data as long as the byte range has been locked by the client; otherwise they too must be sent to the server.

When a client opens a file, it may request that the server grant it an exclusive or batch oplock on the file. The response from the server indicates the type of oplock granted to the client. If cached or read-ahead information was retained after the file was last closed, the client must verify that the last modified time is unchanged when the file is reopened before using the retained information.

The SMB_COM_LOCKING_ANDX SMB is used to convey oplock break requests and acknowledgements (as well as lock and unlock requests).

2.6.1 Level II Oplocks

The Level II oplock protocol is:

```
   Client                     <->  Server

   A            B
   ===========  ===========  ==== ====================================

   Open("foo")               ->
                             <-   Open OK.  Exclusive oplock granted.
   Read                      ->
                             <-   data
                Open("foo")  ->
                             <-   Break to Level II oplock to A
   lock(s)                   ->
                             <-   lock(s) response(s)
   oplock ack                ->
                             <-   Open OK.  Oplock II oplock granted
                                   to B
```

When a client opens a file, it may request an exclusive or batch oplock. If the requested oplock cannot be granted, then the server MAY grant a Level II oplock if the file currently has an oplock on it. If there is currently an exclusive or batch oplock on the file, it must be broken and the break acknowledged before the open is processed. If there is currently a Level II oplock on the file, it does not need to be broken, and the open may be processed immediately.

If any client sends a request to modify the state of a file that has a Level II oplock, the server must ask all clients holding an oplock on the file to break it, but need not wait for an acknowledgement.

2.6.2 Exclusive Oplocks

The exclusive oplock protocol is:

```
   Client                       <-> Server

   A               B
   ==============  ===========  === ================================

   Open ("foo")                 ->
                                <-  Open OK.  Exclusive oplock
                                     granted.
   <locks,
   writes>
   read (large)                 ->
                                <-  read data
   <reads from
   read-ahead >
                   Open("foo")  ->
                                <-  oplock break to A
   lock(s)                      ->
                                <-  lock(s) response(s)
   write(s)                     ->
                                <-  write(s) response(s)
   close or                     ->
   oplock ack
                                <-  open response to B
```

When client A opens the file, it can request an exclusive oplock. Provided no one else has the file open on the server, then the server MAY grant the oplock to client A.

If, at some point in the future, another client, such as client B, requests an open of the same file, or requests a path name based operation on the file, then the server MUST tell client A to relinquish its exclusive oplock. If client B's request will not modify the state of the file, the server MAY tell client A that its exclusive oplock has been replaced by a level II oplock.

When a client's exclusive oplock is broken, it must synchronize the server to the local state of the file (contents and attributes) and any locks it holds on the file, and then acknowledge the oplock break request. After the server receives the acknowledgement, if can process

B's request.

2.6.3 Batch Oplocks

The batch oplock protocol is:

```
Client                     <->  Server

A            B
===========  ============ ==== ===============================

Open("foo")                ->
                           <-   Open OK.  Batch oplock granted.
Read                       ->
                           <-   read data
<close>
<open>
<seek>
read                       ->
                           <-   data
<close>
             Open("foo")   ->
                           <-   Oplock break to A
Close                      ->
                           <-   Close OK to A
                           <-   Open OK to B
```

When client A opens the file, it can request a batch oplock. Provided no one else has the file open on the server, then the server MAY grant the oplock to client A.

If, at some point in the future, another client, such as client B, requests any operation on the same file, then the server MUST tell client A to relinquish its batch oplock. If client B's request will not modify the state of the file (or rename it), the server MAY tell client A that its batch oplock has been replaced by a level II oplock.

If A has the file open at the time the oplock break request is received, its actions will be the same as if it had an exclusive oplock. If A does not have the file open at the time the oplock break request is received, it sends a close to the server. Once the file is actually closed at the server, client B's open request can be processed.

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

The information on authentication that was in previous revisions of this document has been moved to a different specification.

2.9 Distributed Filesystem (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 resolved 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 specification 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 specification 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.

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 Reserved;          // reserved for obsolescent
requests
            UCHAR SecuritySignature[8];   // reserved for MIC
       } Extra;
    };
    USHORT Tid;                       // Tree identifier
    USHORT Pid;                       // 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 specification have identical format up to the ParameterWords fields. (Some obsolescent ones do not.) Different SMBs have a different number and interpretation of ParameterWords and Buffer. All reserved fields in the SMB header must be zero.

3.2.1C   ommand 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	Name: SMB_FLAGS_	Meaning	First Used
0		**Reserved for obsolescent** requests. (LOCK_AND_READ, WRITE_AND_CLOSE)	LANMAN1.0
1		**Reserved (must be zero).**	
2		**Reserved (must be zero).**	
3	**CASELESS**	When on, all pathnames in this SMB must be treated as case-less. When off, the pathnames are case sensitive.	LANMAN1.0
4		**Reserved (clients must send as zero; servers must ignore).**	
5		**Reserved for obsolescent** requests. (SMB_COM_OPEN, SMB_COM_CREATE and SMB_COM_CREATE_NEW)	LANMAN1.0
6		**Reserved for obsolescent** requests. (SMB_COM_OPEN, SMB_COM_CREATE and SMB_COM_CREATE_NEW)	LANMAN1.0
7	**SERVER_RESP**	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 six 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.

Value	Name: SMB_FLAGS2_	Meaning	First Used
0x0001	KNOWS_LONG_NAMES	If set in a request, the server may return long components in path names in the response.	LM1.2X002
0x0002	KNOWS_EAS	If set, the client is aware of extended attributes (EAs).	
0x0004	SECURITY_SIGNATURE	If set, the SMB is integrity checked	
0x0008	RESERVED1	Reserved for future use	
0x0040	IS_LONG_NAME	If set, any path name in the request is a long name	
0x0800	EXT_SEC	If set, the client is aware of Extended Security negotiation	NT LM 0.12
0x1000	DFS	If set, any request pathnames in this SMB should be resolved in the Distributed File System.	NT LM 0.12
0x2000	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.	
0x4000	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	NT LM 0.12

	passing NT status codes is negotiated, this flag should be set for every SMB.	
0x8000 UNICODE	If set, any fields of	NT LM 0.12

datatype STRING in this SMB
message are encoded as
UNICODE. Otherwise, they
are in ASCII.

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

The Pid field identifies to the server the "process" that opened a file (see SMB_COM_FLUSH) or that owns a byte range lock (see SMB_COM_LOCKING_ANDX). This "process" may or may not correspond to the client operating system's notion of process.

The client chooses the value of the Pid field; servers MUST set the Pid field of responses to the same value as in the corresponding request. The Pid is relative to a transport connection -- the same Pid in requests sent over different connections will be considered to represent a different process.

3.2.6 Uid Field

Uid is a user ID 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 that a single user 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 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 at one time. Servers MAY respond to requests in any order, but a response message MUST always contain the same Mid value as the corresponding request message. The client MUST NOT have multiple outstanding requests to a server with the same Mid.

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". But some SMB requests do indicate timeout periods for the completion of the request on the server. 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:

- o The protocol strings in the Negotiate SMB request.

- o 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 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.

3.4 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.

If the client is using 8.3 names, each part of the name (base (8) or extension (3)) is treated separately. For long filenames the . in the name is significant even though there is no longer a restriction on the size of each of the components.

The ? character is a wild card for a single character. If a 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

The * character matches an entire part of the name, as does an empty specification for that part. A part consisting of * means that the rest of the component should be filled with ? and the search should be performed with this wildcard character. For example, "*.abc" or ".abc" match any file with an extension of "abc". "*.*", "*" or "null" match 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.

3.5 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.

3.6 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, where the base time (the 64-bit integer with value 0) is the beginning of the year 1601 AD in the Gregorian calendar. 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.

3.7 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
```

3.8 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	Meaning
0 - 15	Specific rights. Contains the access mask specific to the object type associated with the mask.
16 - 23	Standard rights. Contains the object's standard access rights and can be a combination of the following predefined flags:

Bit	Flag	Meaning
16	DELETE	Delete access
17	READ_CONTROL	Read access to the owner, group, and discretionary access-control list (ACL) of the security descriptor
18	WRITE_DAC	Write access to the discretionary access-control list (ACL)
19	WRITE_OWNER	Write access to owner
20	SYNCHRONIZE	Windows NT: Synchronize access

Bits	Meaning
24	Access system security (ACCESS_SYSTEM_SECURITY). 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.
25	Maximum allowed (MAXIMUM_ALLOWED)
26, 27	Reserved
28	Generic all (GENERIC_ALL)
29	Generic execute (GENERIC_EXECUTE)
30	Generic write (GENERIC_WRITE)
31	Generic read (GENERIC_READ)

3.9 Open Function Encoding

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

bits:

```
        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.
```

3.10 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:

bits:

```
    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.
```

3.11 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

3.12 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.
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.
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.)

3.13 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:

- o The embedded command does not repeat the SMB header information. Rather the next SMB starts at the WordCount field.

- o All multiple (chained) requests must fit within the negotiated transmit size. For example, if SMB_COM_TREE_CONNECT_ANDX included OPENandX SMB_COM_OPEN_ANDX which included SMB_COM_WRITE were sent, they would all have to fit within the negotiated buffer size. This would limit the size of the write.

- o 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.

- o 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.

- o 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.

3.14 "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).

3.14.1 SMB_COM_TRANSACTION2 Format

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 - also disconnect TID in 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)
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[ParameterCount];	Parameter bytes (# = ParameterCount)
UCHAR Pad1[];	Pad to SHORT or LONG
UCHAR Data[DataCount];	Data bytes (# = DataCount)

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

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[ParameterCount];	Parameter bytes (# = ParameterCount)
UCHAR Pad1[];	Pad to SHORT or LONG
UCHAR Data[DataCount];	Data bytes (# = DataCount)

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

3.14.2 **3.13.2** SMB_COM_NT_TRANSACTION Formats

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                            Parameter bytes
Parameters[ParameterCount];
UCHAR Pad2[];                    Pad to LONG
UCHAR Data[DataCount];   Data
bytes
```

```
Interim Server Response          Description
===============================  ====================================

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

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

UCHAR WordCount;                 Count of parameter words = 18
UCHAR Reserved[3];               MBZ
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                            Parameter bytes
Parameters[ParameterCount];
UCHAR Pad2[];                    Pad to LONG
UCHAR Data[DataCount];           Data bytes
```

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[ParameterCount];	Parameter bytes
UCHAR Pad2[];	Pad to SHORT or LONG
UCHAR Data[DataCount];	Data bytes

3.14.3 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 the server may return (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 does 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 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_CLOSE_PRINT_FILE
SMB_COM_WRITE_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
```

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

```
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 that it can communicate with. 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 ChallengeLength;	Length of Challenge; MBZ if not LM2.1 dialect or later
USHORT Reserved;	MBZ
USHORT ByteCount	Count of data bytes
UCHAR Challenge[];	The challenge
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 CIFS Security document.

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 = use challenge/response authentication bit 2: 1 = Security Signatures (SMB integrity check) enabled bit 3: 1 = Security Signatures (SMB integrity check) 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)
UCHAR SecurityBlobLength;	Length of SecurityBlob
USHORT ByteCount;	Count of data bytes
UCHAR GUID[16]	A globally unique identifier assigned to the server; present only when CAP_EXTENDED_SECURITY is on in the 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.
UCHAR OemDomainName[];	The name of the domain (in OEM chars); not present if CAP_EXTENDED_SECURITY is on in the Capabilities field

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.

Capabilities allows the server to tell the client what it supports. 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
CAP_LOCK_AND_READ	0x0100	The server supports the SMB_COM_LOCK_AND_READ SMB
CAP_NT_FIND	0x0200	
CAP_DFS	0x1000	The server is DFS aware
CAP_LARGE_READX	0x4000	The server supports large SMB_COM_READ_ANDX
CAP_LARGE_WRITEX	0x8000	The server supports large SMB_COM_READ_ANDX
CAP_RESERVED	0x02000000	Reserved for future use.
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 provides 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.

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

and the 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 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_AND X 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

The 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 SecurityBlobLength	length of Security Blob that follows in a later field
USHORT ByteCount;	Count of data bytes
UCHAR SecurityBlob[]	SecurityBlob of length specified in 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_REQUEIRED (a value of 0xC0000016) in the SMB status. The client can then repeat the SessionSetupAndX SMB with the next 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 CaseInsensitivePasswordLength;	Account password size, ANSI
USHORT CaseSensitivePasswordLength;	Account password size, Unicode
ULONG Reserved;	must be 0
ULONG Capabilities;	Client capabilities
USHORT ByteCount;	Count of data bytes; min = 0
UCHAR CaseInsensitivePassword[];	Account Password, ANSI
UCHAR CaseSensitivePassword[];	Account Password, Unicode
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:

Capability Name	Encoding	Description
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_NT_FIND	0x0200	
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

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 one 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

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

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 specific 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
USHORT ByteCount;	Count of data bytes; min = 3
UCHAR Service[];	Service type connected to. Always ANSII.
STRING NativeFileSystem[];	Native file system for this tree

NativeFileSystem is the name of the filesystem; values to be expected include FAT, NTFS, etc.

OptionalSupport bits has the encoding:

Name	Encoding	Description
SMB_SUPPORT_SEARCH_BITS	0x0001	
SMB_SHARE_IS_IN_DFS	0x0002	

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

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 to with 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.

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

Parameter Block Encoding	Description
USHORT Information Level;	Level of information requested

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 calls to 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.

InformationLevel	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

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

4.1.6.1 SMB_INFO_ALLOCATION

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

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

Data Block Encoding	Description

LARGE_INTEGER	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

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

Data Block Encoding	Value
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 over the network. Some may change to be accessible over the network. The values for device types that may never be accessible over the network may be redefined to be just reserved at some date in the future.

Characteristics 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

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 is the sum of any of the following:

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

4.1.6.7 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 the right 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.

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

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.

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.

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
LARGE_INTEGER AllocationSize;	Initial allocation size
ULONG ExtFileAttributes;	File attributes
ULONG ShareAccess;	Type of share access
ULONG CreateDisposition;	Action to take if file exists or not
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 DesiredAccess parameter is specified in section 3.8 on Access Mask Encoding.

If no value is specified, it still allows an application to 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 on Extended File Attribute Encoding

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

Name	Value	Meaning
	0	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:

CREATE_NEW	Creates a new file. The function fails if the specified file already exists.
CREATE_ALWAYS	Creates a new file. The function overwrites the file if it exists.
OPEN_EXISTING	Opens the file. The function fails if the file does not exist.
OPEN_ALWAYS	Opens the file, if it exists. If the file does not exist, act like CREATE_NEW.
TRUNCATE_EXISTING	Opens the file. Once opened, the file is truncated so that its size is zero bytes. The calling process must open the file with at least GENERIC_WRITE access. The function fails if the file does not exist.

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

SECURITY_ANONYMOUS	Specifies to impersonate the client at the Anonymous impersonation level.
SECURITY_IDENTIFICATION	Specifies to impersonate the client at the Identification impersonation level.
SECURITY_IMPERSONATION	Specifies to impersonate the client at the Impersonation impersonation level.
SECURITY_DELEGATION	Specifies to impersonate the client at the Delegation impersonation level.

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

SECURITY_CONTEXT_TRACKING Specifies that the security tracking mode is dynamic. If this flag is not specified, Security Tracking Mode is static.

SECURITY_EFFECTIVE_ONLY Specifies that only the enabled aspects of the client's security context are available to the server. If you do not specify this flag, 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 response is as follows:

Server Response	Description
UCHAR WordCount;	Count of parameter words = 26
UCHAR AndXCommand;	0xFF = None
UCHAR AndXReserved;	MBZ
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

4.2.2 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.

Request Parameter Block Encoding	Description
ULONG Flags;	Creation flags (see below)
ULONG RootDirectoryFid;	Optional directory for relative open
ACCESS_MASK DesiredAccess;	Desired access
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 to take if file exists or not
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)

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

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

Output 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 for the definition of the parameters.

4.2.3 CREATE_TEMPORARY: Create Temporary File

The server creates a data file in Directory relative to Tid in the SMB header and assigns a unique name to it.

Client Request	Server Response
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 which 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.

4.2.4 **READ_ANDX:** Read Bytes

Large File 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 MBZ
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

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 MBZ
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 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 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 MBZ
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

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_LOCK_AND_READ
SMB_COM_CLOSE
SMB_COM_READ_ANDX
SMB_COM_WRITE_ANDX

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).

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;	Num. unlock range structs following
USHORT NumberOfLocks;	Num. lock range structs following
USHORT ByteCount;	Count of data bytes
LOCKING_ANDX_RANGE Unlocks[];	Unlock ranges
LOCKING_ANDX_RANGE Locks[];	Lock ranges

LockType 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

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

Large File LOCKING_ANDX_RANGE Format	
USHORT Pid;	PID of process "owning" lock

```
 USHORT Pad;                        Pad to DWORD align (mbz)
 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)
```

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 synchronizing processes' read/write accesses to regions of a file. The locked regions can be anywhere in the logical file. Locking beyond end-of-file is permitted. Any request coming in on the same connection and using the same Pid and Fid as specified in a successful lock request has access to the locked bytes; other requests will be denied the locking, reading, or writing of the locked bytes if they are incompatible with the lock mode.

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. 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 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.)

4.2.6.1 Oplocks

Oplocks are described in the "Opportunistic Locks" section elsewhere in this document. Part of their specification requires that the client will be notified when another client makes certain requests. When that happens, the server delays the second request and notifies the client via an SMB_LOCKING_ANDX SMB 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.

If an acknowledgement is required, the client responds 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. If the client sends an SMB_LOCKING_ANDX SMB with the LOCKING_ANDX_OPLOCK_RELEASE flag set and NumberOfLocks is zero, the server MUST NOT send a response. 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 which it does not have open, that notification should be ignored.

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_WRITE
SMB_COM_FLUSH
SMB_COM_READ_ANDX
SMB_COM_WRITE_ANDX

4.2.6.2 Errors

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

4.2.7 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

4.2.7.1 Errors

ERRDOS/ERRbadfid
ERRSRV/ERRinvid
ERRSRV/ERRbaduid

4.2.8 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
```

4.2.8.1 Errors

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

4.2.9 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. Attributes are described in the "Attribute Encoding" section 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 much 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

4.2.9.1 Errors

ERRDOS/ERRbadpath
ERRDOS/ERRbadfile

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

4.2.10 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

Files 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

4.2.10.1 Errors

ERRDOS/ERRbadpath
ERRDOS/ERRbadfile
ERRDOS/ERRnoaccess

```
ERRDOS/ERRdiffdevice
ERRHRD/ERRnowrite
ERRSRV/ERRaccess
ERRSRV/ERRinvdevice
ERRSRV/ERRinvid
ERRSRV/ERRbaduid
```

4.2.11 **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.

4.2.11.1 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

4.2.12 COPY: Copy File

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. Flags 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.

4.2.12.1 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

4.2.13 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

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

The following InformationLevels may be requested:

Information Level	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
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

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.

4.2.13.1 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)

4.2.13.2 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

4.2.13.3 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.

4.2.13.4 SMB_QUERY_FILE_BASIC_INFO

Data Block Encoding	Description
LARGE_INTEGER CreationTime;	Time when file was created
LARGE_INTEGER LastAccessTime;	Time of last file access
LARGE_INTEGER LastWriteTime;	Time of last write to the file
LARGE_INTEGER ChangeTime	Time when file was last changed
USHORT Attributes;	File Attributes

4.2.13.5 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

4.2.13.6 SMB_QUERY_FILE_EA_INFO

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

4.2.13.7 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

4.2.13.8 SMB_QUERY_FILE_ALL_INFO

Data Block Encoding	Description
LARGE_INTEGER CreationTime;	Time when file was created
LARGE_INTEGER LastAccessTime;	Time of last file access
LARGE_INTEGER LastWriteTime;	Time of last write to the file
LARGE_INTEGER 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 Index Number	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 Index Number	A file system unique identifier
LARGE_INTEGER CurrentByteOffset	Current byte offset within the file
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 and can have any suitable combination of the following values:

Value		Meaning
ILE_READ_DATA	0x00000001	Data can be read from the file
ILE_WRITE_DATA	0x00000002	Data can be written to the file
ILE_APPEND_DATA	0x00000004	Data can be appended to the file
ILE_READ_EA	0x00000008	Extended attributes associated with the file can be read
ILE_WRITE_EA	0x00000010	Extended attributes associated with the file can be written
ILE_EXECUTE	0x00000020	Data can be read into memory from the file using system paging I/O
ILE_READ_ATTRIBUTES	0x00000080	Attributes associated with the file can be read
ILE_WRITE_ATTRIBUTES	0x00000100	Attributes associated with the file can be written
ELETE	0x00010000	The file can be deleted
EAD_CONTROL	0x00020000	The access control list and ownership associated with the file can be read
RITE_DAC	0x00040000	The access control list and ownership associated with the file can be written.
RITE_OWNER	0x00080000	Ownership information associated with the file can be written
YNCHRONIZE	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:

Value		Meaning
FILE_WRITE_THROUGH	0x00000002	File is opened in mode where data is written to 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:

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

4.2.13.9 SMB_QUERY_FILE_ALT_NAME_INFO

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

4.2.13.10 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 StreamAllocationSize	Allocated size of the stream in number of bytes
STRING FileName	Name of the stream

4.2.13.11 SMB_QUERY_FILE_COMPRESSION_INFO

Data Block Encoding	Description
LARGE_INTEGER CompressedFileSize	Size of the compressed file in 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]	

4.2.14 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.

4.2.15 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:

Information Level	Value
SMB_INFO_STANDARD	1
SMB_INFO_QUERY_EA_SIZE	2
SMB_INFO_QUERY_ALL_EAS	4

The response formats are:

4.2.15.1 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)

4.2.15.2 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

4.2.16 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:

Information Level	Value
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

The two levels below 0x101 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, the transaction response has 1 parameter word which should be ignored by the client.

4.2.16.1 SMB_FILE_DISPOSITION_INFO

Response Field	Value
BOOLEAN	A boolean which is TRUE if the file is marked

FileIsDeleted	for deletion

4.2.16.2 SMB_FILE_ALLOCATION_INFO

Response Field	Value
LARGE_INTEGER	File Allocation size in number of bytes

4.2.16.3 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

4.3 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

4.3.2 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

4.3.3 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. 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 to file name of 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.

InformationLevel Name	Value
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

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.

A four-byte resume key precedes each data item (described below) if bit **2 in the Flags field is set, i.e. if the request indicates the server** should return resume keys.

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 beginning of next one
ULONG FileIndex;	
LARGE_INTEGER CreationTime;	file creation time
LARGE_INTEGER LastAccessTime;	last access time
LARGE_INTEGER LastWriteTime;	last write time
LARGE_INTEGER ChangeTime;	last attribute change time
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 beginning of next one
ULONG FileIndex;	
LARGE_INTEGER CreationTime;	file creation time
LARGE_INTEGER LastAccessTime;	last access time
LARGE_INTEGER LastWriteTime;	last write time
LARGE_INTEGER ChangeTime;	last attribute change time
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 beginning of next one
ULONG FileIndex;	
LARGE_INTEGER CreationTime;	file creation time
LARGE_INTEGER LastAccessTime;	last access time
LARGE_INTEGER LastWriteTime;	last write time
LARGE_INTEGER ChangeTime;	last attribute change time
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;	Files full length name

4.3.4.7 SMB_FIND_FILE_NAMES_INFO

Response Field	Description
ULONG NextEntryOffset;	Offset from this structure to beginning of next one
ULONG FileIndex;	
ULONG FileNameLength;	Length of FileName in bytes
STRING FileName;	Files full length name

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 to file name of last entry, if server needs it to resume search; else 0
UCHAR Data[TotalDataCount]	Level dependent info about the matches found in the search

4.3.6 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;	MBZ

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 an alternate status code 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

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 client
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
UNICODESTRINGE 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 DfsAlternatePathOffset  Offset, in bytes from the beginning of
                                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.

4.4.2 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 to 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 was 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.

4.5 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 io or fs control. Unless BIT0 of ISFLAGS is set.
BOOLEAN IsFsctl;	Indicates whether the command is a device control (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
Data[TotalDataCount]	Passed to the Fsctl or Ioctl

Server Response	Description
SetupCount	1
Setup[0]	Length of information returned by io or fs control
DataCount	Length of information returned by io or fs control
Data[DataCount]	The results of the io or fs control

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;	MBZ
ULONG SecurityInformation;	Fields of descriptor to set

NtQuerySecurityObject() is called, requesting SecurityInformation. The result of the call is returned to the client in the Data part of the transaction response.

4.5.3 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;	MBZ
ULONG SecurityInformation;	Fields of SD that 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.

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_COMPLETE	0x20
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_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
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_NT_TRANSACT	0xA0
SMB_COM_NT_TRANSACT_SECONDARY	0xA1
SMB_COM_NT_CREATE_ANDX	0xA2
SMB_COM_NT_CANCEL	0xA4
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	Description
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
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	0x0D	Create directory with extended attributes
TRANS2_SESSION_SETUP	0x0E	Session setup with extended security information
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:

SubCommand Code	Value	Description
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
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.
DOS LANMAN2.1	DOS LANMAN2.1
LANMAN2.1	OS/2 LANMAN2.1
Windows for Workgroups 3.1a	Windows for Workgroups Version 1.0
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.

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 an 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.

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 @ 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.
ERRinvnid	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.
ERRfilespecs	67	The Fid and pathname parameters contained an invalid combination of values.
ERRbadpermits	69	The access permissions specified for a file or directory are not a valid combination. The server cannot set the requested attribute.
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.
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.

7 Legal Notice

Microsoft does not know of any third-party rights that are violated by this contribution. Microsoft makes no other representations regarding this contribution.

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] 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 Authors' Addresses

Paul Leach
Dilip Naik
Microsoft
1 **Microsoft Way**
Redmond, WA 98052
paulle@microsoft.com
dilipn@microsoft.com

10 Appendix A -- NETBIOS transport over TCP

When operating CIFS over the NETBIOS transport over TCP, connections are established and messages transferred as specified in RFC 1001 and RFC 1002.

Message transport is done using NETBIOS session service (see section 5.3 of RFC 1001 and section 4.3 of RFC 1002).

10.1 Connection Establishment

After the server name has been resolved to an IP address, then a connection to the server needs to be established if one has not already been set up. Connection establishment is done using the NETBIOS session service, which requires the client to provide a "calling name" and a "called 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; the called name is always "*SMBSERVER   ". Connection establishment results in a "Session Request" packet to port 139 **(see** section 4.3.2 of RFC 1002**).**

10.1.1 Backwards compatability

If a CIFS client wishes to inter-operate with older SMB servers, then if the server rejects the session request, it can retry with a new called name. The choice of the new called name depends on the name resolution mechanism used. If DNS was used, the called name should be constructed from the first component of the server's DNS name, truncated to 15 characters if necessary, and then padded to 16 characters with blank (20 hex) characters. If NETBIOS was used, then the called named is just the NETBIOS name. If these fail, then a NETBIOS "Adapter Status" request may be made to obtain the server's NETBIOS name, and the connection establishment retried with that as the called name.

10.2 Server-side Connection Procedures

A CIFS server running over NETBIOS MUST accepts any session request specifying a called name of "*SMBSERVER   ".

In addition, if it wishes to support older SMB clients, it MAY have one or more NETBIOS names and accept session request specifying them as the called name.

11 Appendix B -- TCP transport

When operating CIFS over TCP, connections are established to TCP port TBD, and each message is framed as follows:

```
                        1 1 1 1 1 1 1 1 1 1 2 2 2 2 2 2 2 2 2 2 3 3
    0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
   +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
   |      ZERO     |                    LENGTH                     |
   +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
   |                                                               |
   /               SMB (Packet Type Dependent)                     /
   |                                                               |
   +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

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.