-- --

Microsoft Networks

SMB FILE SHARING PROTOCOL EXTENSIONS

SMB File Sharing Protocol Extensions Version 3.0

Document Version 1.11

June 19, 1990

Microsoft Corporation

1. INTRODUCTION

This document defines extensions to the LANMAN 1.0 Microsoft file sharing protocol as defined in the SMB File Sharing Protocol Extension version 2.0 , document version 3.2, and the OenNet/Microsoft Networks File Sharing Protocol (Intel PN 136329-001) (sometimes referred to as the "core" protocol). These extensions are primarily to provide support for Operating Systems which use installable file systems. The support for installable file systems require that extended attribute data blocks, of potentially greater size than a negotiated buffer, are supplied with requests that could be transported in a negotiated buffer in the LANMAN 1.0 enviroment. The extended file sharing protocol is not intended to be specific to OS/2. It is anticipated that other Operating Systems will have many similar requirements and that they will use the same services and protocols to meet them.

This extension, when combined with the LANMAN 1.0 and core protocol, allows all file oriented OS/2 version 1.2 functions to be performed on remote files using LANMAN 2.0.

The extended protocol defined in this document is selected by the dialect string "LM1.2X002" in the core protocol negotiate request.

Acronyms used include:

VC - Virtual Circuit. A transport level connection (sometimes called a session) between two networked machines (nodes).

TID - Tree Identifier. A token representing an instance of authenticated use of a network resource (often a shared subdirectory tree structure).

UID - User Identifier. A token representing an authenticated user of a network resource.

PID - Process Identifier. A number which uniquely identifies a process on a node.

MID - Multiplex Identifier. A number which uniquely identifies a protocol request and response within a process.

FID - File Identifier. A number which identifies an instance of an open file (sometimes called file handle).

T.B.D.- To Be Defined. Further detail will be provided at a later time.

MBZ - Must Be Zero. All reserved fields must be set to zero by the consumer.

2. MESSAGE FORMAT

All messages sent while using the extended protocol (both the core messages used and the additional messages defined in this document) will have the following format.

```
 BYTE    smb_idf[4];   /* contains 0xFF,'SMB' */
 BYTE    smb_com;      /* command code */
 BYTE    smb_rcls;     /* error class */
 BYTE    smb_reh;      /* reserved for future */
WORD     smb_err;      /* error code */
 BYTE    smb_flg;      /* flags */
WORD     smb_flg2;     /* flags */
WORD     smb_res[6];   /* reserved for future */
WORD     smb_tid;      /* authenticated resource identifier */
WORD     smb_pid;      /* caller's process id */
WORD     smb_uid;      /* authenticated user id */
WORD     smb_mid;      /* multiplex id */
 BYTE    smb_wct;      /* count of 16-bit words that follow */
WORD     smb_vwv[];    /* variable number of 16-bit words */
WORD     smb_bcc;      /* count of bytes that follow */
 BYTE    smb_buf[];    /* variable number of bytes */
```

The structure defined from smb_idf through smb_wct is the fixed portion of the SMB structure sometimes referred to as the SMB header. Following the header there is a variable number of words (defined by smb_wct) and following that is smb_bcc which defines an additional variable number of bytes.

> A BYTE is 8 bits.
> A WORD is two BYTEs.
> The BYTEs within a WORD are ordered such that the low BYTE precedes the high BYTE.
> A DWORD is two WORDs.
> The WORDs within a DWORD are ordered such that the low WORD precedes the high WORD.

smb_com: - command code.

smb_rcls: - error class (see below).

smb_ret: - error returned (see below).

smb_tid: - Used by the server to identify a resource (e.g., a disk sub-tree). (see below)

smb_pid: - caller's process id. Generated by the consumer (redirector) to uniquely identify a process within the consumer's system. A response message will always contain the same value in smb_pid (and smb_mid) as in the corresponding request message.

smb_mid: - this field is used for multiplexing multiple messages on a single Virtual Circuit (VC) normally when multiple requests are from the same process. The PID (in smb_pid) and the MID (in smb_mid) uniquely identify a request and are used by the consumer to correlate incoming responses to previously sent requests.

3. NOTES:

1. smb_flg can have the following values:

bit0 - When set (returned) from the server in the Negotiate response protocol, this bit indicates that the server supports the "sub dialect" consisting of the LockandRead and WriteandUnlock protocols defined in the SMB File Sharing Protocol Extension version 2.0 , document version 3.2

bit1 - When on (on a protocol request being sent to the server), the consumer guarantees that there is a receive buffer posted such that a "Send.No.Ack" can be used by the server to respond to the consumer's request. The LANMAN 2.0 Redirector for OS/2 will not set this bit.

bit2 - Reserved (must be zero).

bit3 - When on, all pathnames in the protocol must be treated as caseless. When off, the pathnames are case sensitive. This allows forwarding of the protocol message on various extended VCs where caseless may not be the norm. The LANMAN 2.0 Redirector for OS/2 will always have this bit on to indicate caseless pathnames.

bit4 - When on (on the Session Setup and X protocol defined later in this document), all paths sent to the server by the consumer are already in the canonicalized format used by OS/2. This means that file/directory names are in upper case, are valid characters and backslashes are used as seperators.

bit5 - When on (on core protocols Open, Create and Make New), this indicates that the consumer is requesting that the file be "opportunisticaly" locked if this process is the only process which has the file open at the time of the open request. If the server "grants" this oplock request, then this bit should remain set in the coresponding response protocol to indicate to the consumer that the oplock request was granted. See the discussion of "oplock" in the sections defining the "Open and X" and "Locking and X" protocols later in this document (this bit has the same function as bit 1 of smb_flags of the "Open and X" protocol).

bit6 - When on (on core protocols Open, Create and Make New), this indicates that the server should notify the consumer on any action which can modify the file (delete, setattrib, rename, etc.). If not set, the server need only notify the consumer on another open request. See the discussion of "oplock" in the sections defining the "Open and X" and "Locking and X" protocols later in this document (this bit has the same function as bit 2 of smb_flags of the "Open and X" protocol).

bit7 - When on, this protocol is being sent from the server in response to a consumer request. The smb_com (command) field usually contains the same value in a protocol request from the consumer to the server as in the matching response from the server to the consumer. This bit unambiguously distinguishes the command request from the command response. On a multiplexed VC on a node where both server and consumer are active, this bit can be used by the node's SMB delivery system to help identify whether this protocol should be routed to a waiting consumer process or to the server.

2. smb_flg2 can have the following values:

bit0 - When set by the consumer, the running application understands OS/2 1.2 style file names.

bit1 - When set by the consumer, the running application understands extended attributes.

bit2 through bit15 - Reserved (MBZ).

3. smb_uid is the user identifier. It is used by the LANMAN 1.0 extended protocol when the server is executing in "user level security mode" to validate access on protocols which reference symbolicly named resources (such as file open). Thus differing users accessing the same TID may be granted differing access to the resources defined by the TID based on smb_uid. The UID is returned by the server via the Session Set Up protocol. This UID must be used in all SMB's following Session Set Up And X.

4. In the LANMAN 2.0 extended protocol environment the TID represents an instance of an authenticated use. This is the result of a successful NET USE to a server using a valid netname and password (if any).

   If the server is executing in a "share level security mode", the tid is the only thing used to allow access to the shared resource. Thus if the user is able to perform a successful NET USE to the server specifying the appropriate netname and passwd (if any) the resource may be accessed according to the access rights associated with the shared resource (same for all who gained access this way).

   If however the server is executing in "user level security mode", access to the resource is based on the UID (validated on the Session Setup protocol) and the TID is NOT associated with access control but rather merely defines the resource (such as the shared directory tree).

   In most SMB protocols, smb_tid must contain a valid TID. Exceptions include prior to getting a TID established including NEGOTIATE, TREE CONNECT, SESS_SETUPandX and TREE_CONNandX protocols. Other exceptions include QUERY_SRV_INFO some forms of the TRANSACTION protocol and ECHO. A NULL TID is defined as 0xFFFF. The server is responsible for enforcing use of a valid TID where appropriate.

5. As in the core, smb_pid uniquely identifies a consumer process. Consumers inform servers of the creation of a new process by simply introducing a new smb_pid value into the dialogue (for new processes).

   In the core protocol however, the "Process Exit" protocol was used to indicate the catastrophic termination of a process (or session). In the single tasking DOS system, it was possible for hard errors to occur causing the destruction of the process with files remaining open. Thus a Process Exit protocol was used for this occurrence to allow the server to close all files opened by that process.

   In the LANMAN 2.0 extended protocol, no "Process Exit" protocol will be sent. The operating system will ensure that the "close Protocol" will be sent when the last process referencing the file closes it. From the server's point of view, there is no concept of FIDs "belonging to" processes. A FID returned by the server to one process may be used by any other process using the same VC and TID. There is no "birth announcement" (no "fork" protocol) sent to the server. It is up to the consumer to ensure only valid processes gain access to FIDs (and TIDs). On TREE DISCONNECT (or when the VC environment is terminated) the server may invalidate any files opened by any process within the VC environment using that TID.

6. Systems using the LANMAN 2.0 extended protocol will typically be multi-tasked and will allow multiple asynchronous input/output requests per task. Therefore a multiplex ID (smb_mid) is used (along with smb_pid) to allow multiplexing the single consumer/server VC among the consumer's multiple processes, threads and requests per thread.

The consumer is responsible for ensuring that every request includes a value in the smb_mid field which will allow the response to be associated with the correct request (at least the smb_pid and smb_mid must uniquely identify the request/response relationship system wide).

The server is responsible for ensuring that every response contains the same smb_mid value (and smb_pid value) as its request. The consumer may then use the smb_mid value (along with smb_pid value) for associating requests and responses and may have up to the negotiated number of requests outstanding at any time on a multiplexed file server VC.

7. The LANMAN 2.0 extended protocol enhances the semantics of the pathname.

   Two special pathname component values —— "." and ".." —— must be recognized. There may be multiple of these components in a path name. They have the standard meanings —— "." points to its own directory, ".." points to its directory's parent.

   Note that it is the server's responsibility to ensure that the ".." can not be used to gain access to files/directories above the "virtual root" as defined by the Tree Connect (TID).

8. The new LANMAN 2.0 extended protocol requests and responses are variable length (as was true in "core"). Thus additional words may be added in the smb_vwv[] area in the future as well as additional bytes added within the smb_buf[] area. Servers must be implemented such that additional fields in either of these areas will not cause the command to fail. If additional fields are encountered which are not recognized by the server's level of SMB implementation, they should be ignored. This allows for future upgrade of the protocol and eliminates the need for "reserved fields".

9. The contents of response parameters is not guaranteed in the case of an error return (any protocol response with an error set in the SMB header may have smb_wct of zero and smb_bcc count of zero).

10. When LANMAN 2.0 extended protocol has been negotiated, the ERRDOS error class has been expanded to include all errors which may be generated by the OS/2 operating system. As such, the error code values defined for error class ERRDOS in this document are a subset of the possible error values. See the OS/2 operating system documentation for the complete set of possible OS/2 (ERRDOS) error codes.

These semantic changes apply to all "core" requests used by the extended protocol. Where there are additional changes, they are documented with the new requests. The server having negotiated LANMAN 2.0 is expected to still support all LANMAN 1.0 and core protocol requests.

The following are the core protocol requests which must still be supported in the LANMAN 2.0 extended protocol without change. See "File Sharing Protocol" Intel Part number 136329-001 for detailed explanation of each protocol request/response.

TREE CONNECT
TREE DISCONNECT
OPEN FILE
CREATE FILE
CLOSE FILE
FLUSH FILE
READ
WRITE
SEEK
CREATE DIRECTORY
DELETE DIRECTORY
DELETE FILE
RENAME FILE
GET FILE ATTRIBUTES
SET FILE ATTRIBUTES
LOCK RECORD
UNLOCK RECORD
CREATE TEMPORARY FILE (no longer used by LANMAN 2.0 Redirector)
PROCESS EXIT (no longer used by LANMAN 2.0 Redirector)
MAKE NEW FILE
CHECK PATH
GET SERVER ATTRIBUTES
NEGOTIATE PROTOCOL (additional fields in response if LANMAN 2.0 negotiated)
FILE SEARCH
CREATE PRINT FILE
CLOSE PRINT FILE
WRITE PRINT FILE
(core Message Commands are also supported)

The following are the LANMAN 1.0 extended protocol requests which must still be supported in the LANMAN 2.0 extended protocol without change. See SMB File Sharing Protocol Extensions Version 2.0, doument version 3.2, for detailed explanation of each protocol request/response.

SESS_SETUPandX (X is another valid protocol request e.g. TREE_CONNandX)
TREE_CONNandX (X is another valid protocol request e.g. OPEN)
OPENandX (X is another valid protocol request e.g. READ)
READandX (X is another valid protocol request e.g. CLOSE)
WRITEandX (X is another valid protocol request e.g. READ)
FIND (matches OS/2 form of FILE SEARCH)
FIND_UNIQUE (matches OS/2 form of FILE SEARCH)
FIND_CLOSE (matches OS/2 form of FILE SEARCH)
READ_BLOCK_RAW (read larger than negotiated buffer size request raw)
READ_BLOCK_MPX (read larger than negotiated buffer size request multiplexed)
WRITE_BLOCK_RAW (write larger than negotiated buffer size request raw)
WRITE_BLOCK_MPX (write larger than negotiated buffer size request multiplexed)
GET_E_FILE_ATTR (accommodate new OS/2 system call)
SET_E_FILE_ATTR (accommodate new OS/2 system call)
LOCKINGandX (accommodate new OS/2 system call)
COPY_FILE (used when both source and target are remote)
MOVE_FILE (used when both source and target are remote)
IOCTL (pass IOCTL request on to server and retrieve results)
TRANSACTION (allows bytes in/out associated with name)
ECHO (echo sent data back)
WRITEandCLOSE (write final bytes then close file)
LOCKandREAD (Lock bytes then Read locked bytes)
WRITEandUnlock (Write bytes then Unlock bytes)

Of the LANMAN 1.0 extended protocols, only the SESS_SETUPandX request and the COPY_FILE SMB have extended features when LANMAN 2.0 protocol has been negotiated. The format of these SMBs, and their functionality when LANMAN 1.0 protocol has been negotiated is compatible with LANMAN 1.0 protocol but will support new features when LANMAN 2.0 protocol is negotiated. The extended features of SESS_SETUPandX and COPY_FILE SMBs are detailed later in this document.

Support of all core requests within the LANMAN 2.0 extended protocol is mandatory. However, the following core requests will no longer be generated by the OS/2 implementation of the redirector when LANMAN 1.0 or LANMAN 2.0 extended protocol has been negotiated.

PROCESS EXIT
CREATE TEMPORARY FILE
CREATE PRINT FILE
CLOSE PRINT FILE
WRITE PRINT FILE

The only protocol format change to a core protocol service is that the response to the negotiate protocol (NEGOTIATE PROTOCOL) will contain additional fields if the "LM1.2X002" string has been selected by the server thus effectively placing the session into LANMAN 2.0 extended protocol. The additional fields returned will be documented in detail later in this document.

All other protocol requests within the LANMAN 2.0 extended protocol have a new command value from that of a similar function in core protocol. Thus the server need not constantly test the protocol version negotiated. The consumer is expected to only submit appropriate requests within the dialect negotiated.

The following are the new LANMAN 2.0 extended protocol requests, each will be defined in detail later in this document.

TRANSACT2	
FIND_CLOSE	
FIND_NOTIFY_CLOSE	(close a notification handle)
USER LOGOFF and X	(logoff a user id)

4. ARCHITECTURAL MODEL

The Network File Access system fundemental architecture for LANMAN 2.0 is unchanged from the LANMAN 1.0 architecure as described in the SMB File Sharing Protocol Extensions Version 2.0, document version 3.2.

5. LANMAN 1.0 SMB EXTENSIONS

This section describes how to negotiate LANMAN 2.0 protocol and modifications to the LANMAN 1.0 SMB extensions which may be used when the LANMAN 2.0 protocol has been negotiated,

5.1. NEGOTIATE

Request Format is unchanged in order to remain compatible with earlier versions and the core protocol.

Enhanced Response Format (returned only when LANMAN 1.0 dialect or higher is selected):

```
BYTE    smb_wct;          /* value = 13 */
WORD    smb_index;        /* index identifying dialect selected */
WORD    smb_secmode;      /* security mode:
                          bit 0 - 1 = User level security, 0 =  Share level security.  */
                          bit 1 - 1 = encrypt passwords , 0 =  do not encrypt passwords */
WORD    smb_maxxmt;       /* max transmit buffer size server supports,  1k min */
WORD    smb_maxmux;       /* max pending multiplexed requests server supports */
WORD    smb_maxvcs;       /* max VCs per server/consumer session supported */
WORD    smb_blkmode;      /* block read/write mode support :
                          bit 0 - Read Block Raw supported (65,535 bytes max).
                          bit 1 - Write Block Raw supported (65,535 bytes max). */
DWORD   smb_sesskey;      /* Session Key (unique token identifying session) */
WORD    smb_srv_time;     /* server's current time (hhhhh mmmmmm xxxxx)
                          where 'xxxxx' is in two second increments */
WORD    smb_srv_date;     /* server's current date (yyyyyyy mmmm ddddd) */
WORD    smb_srv_tzone;    /* server's current time zone */
DWORD   smb_rsvd;         /* reserved */
WORD    smb_bcc;          /* value = (size of smb_cryptkey) */
BYTE    smb_cryptkey[];   /* Key used for password encryption */
```

This protocol function is unchanged from LANMAN 1.0. The following protocol strings are now accepted:

PC NETWORK PROGRAM 1.0
PCLAN1.0
MICROSOFT NETWORKS 1.03
MICROSOFT NETWORKS 3.0
LANMAN1.0
LM1.2X002
DOS LM1.2X002

The meaning of all protocol strings except "LM1.2X002" and "DOS LM1.2X002" are described in earlier documents. Machines running versions of MS-DOS and wishing to use Lanman 2.0 extensions should negotiate the protocol "DOS LM1.2X002". All other workstations using the LANMAN 2.0 extensions should use the protocol string "LM1.2X002".

5.2. SESSION SETUP and X

Request Format:

```
  BYTE    smb_wct;           /* value = 10 */
  BYTE    smb_com2;          /* secondary (X) command, 0xFF = none */
  BYTE    smb_reh2;          /* reserved (must be zero) */
  WORD    smb_off2;          /* offset (from SMB hdr start) to next cmd (@smb_wct) */
  WORD    smb_bufsize;       /* the consumers max buffer size */
  WORD    smb_mpxmax;        /* actual maximum multiplexed pending requests */
  WORD    smb_vc_num;        /* 0 = first (only), non zero - additional VC number */
 DWORD    smb_sesskey;       /* Session Key (valid only if smb_vc_num != 0) */
  WORD    smb_apasslen;      /* size of account password (smb_apasswd) */
  WORD    smb_encryptlen;    /* size of encryption key (smb_encrypt) */
  WORD    smb_encryptoff;    /* offset (from SMB hdr start) to smb_encrypt */
  WORD    smb_bcc;           /* minimum value = 0 */
  BYTE    smb_apasswd[*];    /* account password (* = smb_apasslen value) */
  BYTE    smb_aname[];       /* account name string */
  BYTE    smb_encrypt[*];    /* encryption key. (* = smb_encryptlen value) */
```

Response Format:

```
  BYTE    smb_wct;          /* value = 3 */
  BYTE    smb_com2;         /* secondary (X) command, 0xFF = none */
  BYTE    smb_res2;         /* reserved (pad to word) */
  WORD    smb_off2;         /* offset (from SMB hdr start) to next cmd (@smb_wct) */
  WORD    smb_action;       /* request mode:
                            bit0 = Logged in successfully - BUT as GUEST */
  WORD    smb_bcc;          /* min value = 0 */
  BYTE    smb_encresp[];    /* server response to request encryption key */
```

Service definition:

This protocol function is unchanged from LANMAN 1.0 except that the station establishing the connection may now verify the validity of the server to which the request was made. The LANMAN 2.0 SESS_SETUPandX request uses a reserved DWORD field from the LANMAN 1.0 request to pass the length and offset of an encryption key contained in the data of the request to the server. The server will use the encryption key to format the smb_encresp field of the response protocol. The station may then use this response to validate the server session.

The LANMAN 2.0 SESS_SETUPandX also returns a UID in the smb_uid field. This is a validated UID which must be supplied by the workstation on all subsequent requests to the server.

5.3. COPY

Request Format:

```
  BYTE    smb_wct;           /* value = 3 */
  WORD    smb_tid2;          /* second (destination) path tid */
  WORD    smb_ofun;          /* what to do if destination file exists */
  WORD    smb_flags;         /* flags to control copy operations:
                             bit 0 - destination must be a file.
                             bit 1 - destination must be a directory.
                             bit 2 - copy destination mode: 0 = binary, 1 = ASCII.
                             bit 3 - copy source mode: 0 = binary, 1 = ASCII.
                             bit 4 - verify all writes.
                             bit 5 - tree copy. Source must be a directory.
                                     Copy mode must be binary.
                                     When tree copy is selected smb_cct field in the
                                     response protocol is undefined.
                             bit 6 - Action when source has EA, and dest does not support EAs
                                     0 = Discard EAs, 1 = Fail copy */
  WORD    smb_bcc;           /* minimum value = 2 */
  BYTE    smb_path[];        /* pathname of source file */
  BYTE    smb_new_path[];    /* pathname of destination file */
```

Response Format:

```
  BYTE    smb_wct;        /* value = 1 */
  WORD    smb_cct;        /* number of files copied */
  WORD    smb_bcc;        /* minimum value = 0 */
  BYTE    smb_errfile[];  /* pathname of file where error occured - ASCIIZ */
```

Service:

The COPY protocol function for LANMAN 2.0 is unchanged from LANMAN 1.0 except that the request may now be used to specify a tree copy on the remote server. The tree copy mode is selected by setting bit 5 of the smb_flags word in the COPY request. When the tree copy option is selected the destination must not be an existing file and the source mode must be binary. A request with bit 5 of the smb_flags word set and either bit 0 or bit 3 set is therefore an error. When the tree copy mode is selected the smb_cct word of the response protocol is undefined.

6. EXTENDED PROTOCOL

The format of enhanced and new commands is defined commencing at the smb_wct field. All messages will include the standard SMB header defined in section 1.0. When an error is encountered a server may choose to return only the header portion of the response (i.e., smb_wct and smb_bcc both contain zero).

6.0.1. TRANSACT2

Primary Request Format:

```
  BYTE    smb_wct;        /* value = (14 + value of smb_suwcnt) */
  WORD    smb_tpscnt;     /* total number of parameter bytes being sent */
  WORD    smb_tdscnt;     /* total number of data bytes being sent */
  WORD    smb_mprcnt;     /* max number of parameter bytes to return */
  WORD    smb_mdrcnt;     /* max number of data bytes to return */
  BYTE    smb_msrcnt;     /* max number of setup words to return */
  BYTE    smb_rsvd;       /* reserved (pad above to word) */
  WORD    smb_flags;      /* additional information:
                             bit 0 - if set, also disconnect TID in smb_tid
                             bit 1 - if set, transaction is one way (no final response) */
 DWORD    smb_timeout;    /* number of milliseconds to wait for completion */
  WORD    smb_rsvd1;      /* reserved */
  WORD    smb_pscnt;      /* number of parameter bytes being sent this buffer */
  WORD    smb_psoff;      /* offset (from start of SMB hdr) to parameter bytes */
  WORD    smb_dscnt;      /* number of data bytes being sent this buffer */
  WORD    smb_dsoff;      /* offset (from start of SMB hdr) to data bytes */
  BYTE    smb_suwcnt;     /* set up word count */
  BYTE    smb_rsvd2;      /* reserved (pad above to word) */
  WORD    smb_setup[*];   /* variable number of set up words (* = smb_suwcnt) */
  WORD    smb_bcc;        /* total bytes (including pad bytes) following */
  BYTE    smb_name[1];    /* Must be a null byte */
  BYTE    smb_pad[];      /* (optional) to pad to word or dword boundary */
  BYTE    smb_param[*];   /* param bytes (* = value of smb_pscnt) */
  BYTE    smb_pad1[];     /* (optional) to pad to word or dword boundary */
  BYTE    smb_data[*];    /* data bytes (* = value of smb_dscnt) */
```

Interim Response Format (if no error - ok send remaining data):

```
  BYTE    smb_wct;    /* value = 0 */
  WORD    smb_bcc;    /* value = 0 */
```

Secondary Request Format (more data - may be zero or more of these):

```
  BYTE    smb_wct;        /* value = 9 */
  WORD    smb_tpscnt;     /* total number of parameter bytes being sent */
  WORD    smb_tdscnt;     /* total number of data bytes being sent */
  WORD    smb_pscnt;      /* number of parameter bytes being sent this buffer */
  WORD    smb_psoff;      /* offset (from start of SMB hdr) to parameter bytes */
  WORD    smb_psdisp;     /* byte displacement for these parameter bytes */
  WORD    smb_dscnt;      /* number of data bytes being sent this buffer */
  WORD    smb_dsoff;      /* offset (from start of SMB hdr) to data bytes */
  WORD    smb_dsdisp;     /* byte displacement for these data bytes */
  WORD    smb_fid;        /* file id for handle based requests, else 0xffff */
  WORD    smb_bcc;        /* total bytes (including pad bytes) following */
  BYTE    smb_pad[];      /* (optional) to pad to word or dword boundary */
  BYTE    smb_param[*];   /* param bytes (* = value of smb_pscnt) */
  BYTE    smb_pad1[];     /* (optional) to pad to word or dword boundary */
  BYTE    smb_data[*];    /* data bytes (* = value of smb_dscnt) */
```

Response Format (may respond with zero or more of these):

```
 BYTE   smb_wct;        /* value = 10 + value of smb_suwcnt */
 WORD   smb_tprcnt;     /* total number of parameter bytes being returned */
 WORD   smb_tdrcnt;     /* total number of data bytes being returned */
 WORD   smb_rsvd;       /* reserved */
 WORD   smb_prcnt;      /* number of parameter bytes being returned this buf */
 WORD   smb_proff;      /* offset (from start of SMB hdr) to parameter bytes */
 WORD   smb_prdisp;     /* byte displacement for these parameter bytes */
 WORD   smb_drcnt;      /* number of data bytes being returned this buffer */
 WORD   smb_droff;      /* offset (from start of SMB hdr) to data bytes */
 WORD   smb_drdisp;     /* byte displacement for these data bytes */
 BYTE   smb_suwcnt;     /* set up return word count */
 BYTE   smb_rsvd1;      /* reserved (pad above to word) */
 WORD   smb_setup[*];   /* variable # of set up return words (* = smb_suwcnt) */
 WORD   smb_bcc;        /* total bytes (including pad bytes) following */
 BYTE   smb_pad[];      /* (optional) to pad to word or dword boundary */
 BYTE   smb_param[*];   /* param bytes (* = value of smb_prcnt) */
 BYTE   smb_pad1[];     /* (optional) to pad to word or dword boundary */
 BYTE   smb_data[*];    /* data bytes (* = value of smb_drcnt) */
```

Service:

The Transaction2 protocol allows transfer of parameter and data blocks greater than a negotiated buffer size between the requester and the server.

The Transaction2 command scope includes (but is not limited to) IOCTL device requests and file system requests which require the transfer of an extended attribute list.

The Transaction2 protocol is used to transer a request for any of a set of supported functions on the server which may require the transfer of large data blocks. The function requested is identified by the first word in the transaction2 smb_setup field. Other function specific information may follow the function identifier in the smb_setup file id or in the smb_param filed. The functions supported are not defined by the protocol, but by consumer/server implementations. The protocol simply provides a means of delivering them and retrieving the results.

The number of bytes needed in order to perform the TRANSACT2 request may be more than will fit in a single buffer.

At the time of the request, the consumer knows the number of parameter and data bytes expected to be sent and passes this information to the server via the primary request (smb_tpscnt and smb_tdscnt). This may be reduced by lowering the total number of bytes expected (smb_tpscnt and/or smbtdscnt) in each (any) secondary request.

Thus when the amount of parameter bytes received (total of each smb_pscnt) equals the total amount of parameter bytes expected (smallest smb_tpscnt) received, then the server has received all the parameter bytes.

Likewise, when the amount of data bytes received (total of each smb_dscnt) equals the total amount of data bytes expected (smallest smb_tdscnt) 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 (smb_psoff and smb_dsoff) and the length fields (smb_pscnt and smb_dscnt). The displacement of the bytes (relative to start of each) is also known (smb_psdisp and smb_dsdisp). Thus the server is able to reasemble the parameter and data bytes should the "packets" (buffers) 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 Consumer knows the maximum amount of data bytes and parameter bytes which the server may return (from smb_mprcnt and smb_mdrcnt of the request). Thus it initializes its bytes expected variables to these values. The Server then informs the consumer of the actual amounts being returned via each "packet" (buffer) of the response (smb_tprcnt and smb_tdrcnt).

The server may reduce the expected bytes by lowering the total number of bytes expected (smb_tprcnt and/or smb_tdrcnt) in each (any) response.

Thus when the amount of parameter bytes received (total of each smb_prcnt) equals the total amount of parameter bytes expected (smallest smb_tprcnt) received, then the consumer has received all the parameter bytes.

Likewise, when the amount of data bytes received (total of each smb_drcnt) equals the total amount of data bytes expected (smallest smb_tdrcnt) received, then the consumer has received all the data bytes.

The parameter bytes should normally be returned first followed by the data bytes. However, the consumer knows where each begins and ends in each buffer by the offset fields (smb_proff and smb_droff) and the length fields (smb_prcnt and smb_drcnt). The displacement of the bytes (relative to start of each) is also known (smb_prdisp and smb_drdisp). Thus the consumer is able to reasemble the parameter and data bytes should the "packets" (buffers) be received out of sequence.

Thus the flow is:

1 The consumer sends the first (primary) request which identifies 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 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 TRANSACT2 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 consumer then sends another buffer full of bytes to the server. On each iteration of this secondary request, smb_tpscnt and/or smb_tdscnt could be reduced. This step is repeated until all bytes have been delivered to the server (total of all smb_pscnt equals smallest smb_tpscnt and total of all smb_dscnt equals smallest smb_tdscnt).

4 The Server sets up and performs the TRANSACT2 with the information provided.

5 Upon completion of the TRANSACT2, 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. On each iteration of this response, smb_tprcnt and/or smb_tdrcnt could be reduced. This step is repeated until all bytes have been delivered to the consumer (total of all smb_prcnt equals smallest smb_tprcnt and total of all smb_drcnt equals smallest smb_tdrcnt).

Thus the flow is:

1 The consumer sends the first (primary) request which identifies the total bytes (parameters and data) which are to be sent, 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 TRANSACT2 completion. The parameter bytes are immediately followed by the data bytes (the length fields identify the break point). 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 consumer then sends another buffer full of bytes to the server. This step is repeated until all bytes have been delivered to the server.

4 The Server sets up and performs the TRANSACT2 with the information provided.

5 Upon completion of the TRANSACT2, the server sends back up to the 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 bytes requested have been returned. On each iteration of this response, smb_rprcnt and smb_rdrcnt are reduced by the number of matching bytes returned in the previous response. The parameter count (smb_rprcnt) is expected to go to zero first because the parameters are sent before the data. The data count (smb_rdrcnt) may then continue to be counted down. Fewer than the requested number of bytes may be returned.

The flow for the TRANSACT2 protocol when the request parameters and data does NOT all fit in a single buffer is:

```
consumer  ----------> TRANSACT2 request (data) >-------------> server
consumer <-----------< OK send remaining data <  -------------  server
consumer  ------> TRANSACT2 secondary request 1 (data) >--------> server
consumer  ------> TRANSACT2 secondary request 2 (data) >--------> server
.                              .                                  .
consumer  ------> TRANSACT2 secondary request n (data) >--------> server
.                              .                                  .
.              (server sets up and performs the TRANSACT2)        .
.                              .                                  .
consumer <  --------< TRANSACT2 response 1 (data) <  ----------- server
consumer <  --------< TRANSACT2 response 2 (data) <  ----------- server
.                              .                                  .
consumer <  --------< TRANSACT2 response n (data) <  ----------- server
```

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

```
consumer  ----------> TRANSACT2 request (data) >-------------> server
.                              .                                  .
.              (server sets up and performs the TRANSACT2)        .
.                              .                                  .
consumer <  --------< TRANSACT2 response 1 (data) <  ----------- server
.                   (only one if all data fit in buffer)          .
consumer <  --------< TRANSACT2 response 2 (data) <  ----------- server
.                              .                                  .
consumer <  --------< TRANSACT2 response n (data) <  ----------- server
```

Note that the primary request through the final response make up the complete protocol, thus the TID, PID, UID and MID are expected to remain constant and can be used by both the server and consumer to route the individual messages of the protocol to the correct process.

Transaction may generate the following errors:

Error Class ERRDOS:

ERRnoaccess
ERRbadaccess

Error Class ERRSRV:

ERRerror
ERRinvnid
ERRaccess
ERRmoredata
<implementation specific>

Error Class ERRHRD:

<implementation specific>

6.0.1.1. Defined Transaction2 Protocols

This section specifies some of the defined usages of the Transaction2 protocol. Each of the usages here utilize the basic (and flexible) transaction protocol format. This is NOT meant to be an exhaustive list.

The following function codes are transferred in smb_setup[0] and are used by the server to identify the specific function required.

TRANSACT2_OPEN	0
TRANSACT2_FINDFIRST	1
TRANSACT2_FINDNEXT	2
TRANSACT2_QFSINFO	3
TRANSACT2_SETFSINFO	4
TRANSACT2_QPATHINFO	5
TRANSACT2_SETPATHINFO	6
TRANSACT2_QFILEINFO	7
TRANSACT2_SETFILEINFO	8
TRANSACT2_FSCTL	9
TRANSACT2_IOCTL	10
TRANSACT2_FINDNOTIFYFIRST	11
TRANSACT2_FINDNOTIFYNEXT	12
TRANSACT2_MKDIR	13

6.0.1.1.1. TRANSACT2_OPEN

The function code TRANSACT2_OPEN in smb_setup[0] in the primary TRANSACT2 requests identifies a request to create a file with extended attributes.

Primary Request Format:

```
  BYTE    smb_wct;          /* value = 15 */
  WORD    smb_tpscnt;       /* value = total number of param bytes being sent */
  WORD    smb_tdscnt;       /* total size of extended attribute list */
  WORD    smb_mprcnt;       /* value = maximum return parameter length */
  WORD    smb_mdrcnt;       /* value = 0. No data returned */
  BYTE    smb_msrcnt;       /* value = 0. No setup words to return */
  BYTE    smb_rsvd;         /* reserved (pad above to word) */
  WORD    smb_flags;        /* additional information:
                               bit 0 - 0
                               bit 1 - 0 */
 DWORD    smb_timeout;      /* max milliseconds to wait for resource to open */
  WORD    smb_rsvd1;        /* reserved */
  WORD    smb_pscnt;        /* value = tpscnt, parms must be in primary request */
  WORD    smb_psoff;        /* offset (from start of SMB hdr to parameter bytes */
  WORD    smb_dscnt;        /* number of data bytes being sent this buffer */
  WORD    smb_dsoff;        /* offset (from start of SMB hdr) to data bytes */
  BYTE    smb_suwcnt;       /* value = 1 */
  BYTE    smb_rsvd2;        /* reserved (pad above to word) */
  WORD    smb_setup1;       /* value = 0 :- TRANSACT2_OPEN */
  WORD    smb_bcc;          /* total bytes (including pad bytes) following */
  BYTE    smb_pad[];        /* (optional) to pad to word or dword boundary */
  BYTE    smb_param[*];     /* The parmater block for the the TRANSACT2_OPEN
                             * function is the open specific information in the
                             * following format. */
          WORD              open_flags2;
                                bit 0 - if set, return additional information
                                bit 1 - if set, set single user total file lock

                                bit 2 - if set, the server should notify the consumer
                                        on any action which can modify the file (delete,
                                        setattrib, rename, etc.). if not set, the server
                                        need only notify the consumer on another open
                                        request. */
                                bit 3 - if set, return total length of EAs for the file
           WORD             open_mode;        /* file open mode */
           WORD             open_sattr;      /* search attributes  */
           WORD             open_attr;       /* file attributes (for create) */
          DWORD             open_time;        /* create time */
           WORD             open_ofun;        /* open function */
          DWORD             open_size;        /* bytes to reserve on "create"
                                               * or "truncate" */
           WORD             open_rsvd[5];     /* reserved (must be zero) */
           BYTE             open_pathname[]; /* file pathname */

  BYTE    smb_pad1[];       /* (optional) to pad to word or dword boundary */
  BYTE    smb_data[*];      /* FEAList structure for the file to be openned */
```

Secondary Request Format (more data - may be zero or more of these):

```
  BYTE    smb_wct;        /* value = 9 */
  WORD    smb_tpscnt;     /* total number of parameter bytes being sent */
  WORD    smb_tdscnt;     /* total number of data bytes being sent */
  WORD    smb_pscnt;      /* value = 0. All params in primary request */
  WORD    smb_psoff;      /* value = 0. No parameters in secondary request. */
  WORD    smb_psdisp;     /* value = 0. No parameters in secondary request. */
  WORD    smb_dscnt;      /* number of data bytes being sent this buffer */
  WORD    smb_dsoff;      /* offset (from start of SMB hdr) to data bytes */
  WORD    smb_dsdisp;     /* byte displacement for these data bytes */
  WORD    smb_fid;        /* value = 0xffff, no handle on request */
  WORD    smb_bcc;        /* total bytes (including pad bytes) following */
  BYTE    smb_pad[];      /* (optional) to pad to word or dword boundary */
  BYTE    smb_data[*];    /* data bytes (* = value of smb_dscnt) */
```

Response Format (one only):

```
  BYTE    smb_wct;          /* value = 10 */
  WORD    smb_tprcnt;       /* total parameter length retuned */
  WORD    smb_tdrcnt;       /* value = 0 no data bytes  */
  WORD    smb_rsvd;         /* reserved */
  WORD    smb_prcnt;        /* parameter bytes being returned */
  WORD    smb_proff;        /* offset (from start of SMB hdr) to parameter bytes */
  WORD    smb_prdisp;       /* value = 0 byte displacement for these param bytes */
  WORD    smb_drcnt;        /* value = 0 no data bytes */
  WORD    smb_droff;        /* value = 0 no data bytes */
  WORD    smb_drdisp;       /* value = 0 no data bytes */
  BYTE    smb_suwcnt;       /* value = 0 no set up return words */
  BYTE    smb_rsvd1;        /* reserved (pad above to word) */
  WORD    smb_bcc;          /* total bytes (including pad bytes) following */
  BYTE    smb_pad[];        /* (optional) to pad to word or dword boundary */
  BYTE    smb_param[*];     /* The parmater block for the the TRANSACT2_OPEN
                             * function response is the open specific return
                             * information in the following format. */
            WORD            open_fid;          /* file handle */
           +WORD            open_attribute;    /* attributes of file or device */
          +DWORD            open_time;         /* last modification time */
          +DWORD            open_size;         /* current file size */
           +WORD            open_access;       /* access permissions actually
                                                * allowed */
           +WORD            open_type;         /* file type */
           +WORD            open_state;        /* state of IPC device (e.g. pipe) */
            WORD            open_action;       /* action taken */
           DWORD            open_fileid;       /* server unique file id */
            WORD            open_offerror;     /* offset into FEAList data of first
                                                * error which occured while setting
                                                * the extended attributes. */
         ++DWORD            open_EAlength;     /* Total EA length for opened file */
```

+returned only if bit 0 of open_flags2 is set in primary request
++returned only if bit 3 of open_flags2 is set in primary request

6.0.1.1.2. TRANSACT2_FINDFIRST

The function code TRANSACT2_FINDFIRST in smb_setup[0] in the primary TRANSACT2 request identifies a request to find the first file that matches the specified file specification.

Primary Request Format:

```
  BYTE    smb_wct;          /* value = 15 */
  WORD    smb_tpscnt;       /* value = total number of param bytes being sent */
  WORD    smb_tdscnt;       /* total size of extended attribute list */
  WORD    smb_mprcnt;       /* value = maximum return parameter length */
  WORD    smb_mdrcnt;       /* value = maximum return data length */
  BYTE    smb_msrcnt;       /* value = 0. No setup words to return */
  BYTE    smb_rsvd;         /* reserved (pad above to word) */
  WORD    smb_flags;        /* additional information:
                                bit 0 - 0
                                bit 1 - 0
 DWORD    smb_timeout;      /* value = 0. Not used for find first */
  WORD    smb_rsvd1;        /* reserved */
  WORD    smb_pscnt;        /* value = tpscnt, parms must be in primary request */
  WORD    smb_psoff;        /* offset (from start of SMB hdr to parameter bytes */
  WORD    smb_dscnt;        /* number of data bytes being sent this buffer */
  WORD    smb_dsoff;        /* offset (from start of SMB hdr) to data bytes */
  BYTE    smb_suwcnt;       /* value = 1 */
  BYTE    smb_rsvd2;        /* reserved (pad above to word) */
  WORD    smb_setup1;       /* value = 1  :- TRANSACT2_FINDFIRST */
  WORD    smb_bcc;          /* total bytes (including pad bytes) following */
  BYTE    smb_pad[];        /* (optional) to pad to word or dword boundary */
  BYTE    smb_param[*];     /* The parmater block for the
                               TRANSACT2_FINDFIRST function is the find
                               first specific information in the
                               following format. */
          WORD              findfirst_Attribute;     /* Search attribute */
          WORD              findfirst_SearchCount;
          WORD              findfirst_flags;         /* find flags */
                                                    /* Bit 0: set - close search after
                                                     *            this request.
                                                     * Bit 1: set - close search if end
                                                     *            of search reached.
                                                     * Bit 2: set - Requester requires
                                                     *            resume key for each
                                                     *            entry found.
                                                     */
          WORD              findfirst_FileInfoLevel;  /* Search level */
          DWORD             findfirst_rsvd;
          BYTE              findfirst_FileName[];
  BYTE    smb_pad1[];       /* (optional) to pad to word or dword boundary */
  BYTE    smb_data[*];      /* Additional FileInfoLevel dependent match
                             * information. For a search requiring extended
                             * attribute matching the data buffer contains
                             * the FEAList data for the seach. */
```


Secondary Request Format (more data - may be zero or more of these):

```
  BYTE   smb_wct;        /* value = 9 */
  WORD   smb_tpscnt;     /* totalnumber of parameter bytes being sent */
  WORD   smb_tdscnt;     /* total number of data bytes being sent */
  WORD   smb_pscnt;      /* value = 0. All params in primary request */
  WORD   smb_psoff;      /* value = 0. No parameters in secondary request. */
  WORD   smb_psdisp;     /* value = 0. No parameters in secondary request. */
  WORD   smb_dscnt;      /* number of data bytes being sent this buffer */
  WORD   smb_dsoff;      /* offset (from start of SMB hdr) to data bytes */
  WORD   smb_dsdisp;     /* byte displacement for these data bytes */
  WORD   smb_fid;        /* value = 0xffff, no handle on request */
  WORD   smb_bcc;        /* total bytes (including pad bytes) following */
  WORD   smb_fid;        /* value = 0xffff, no handle on request */
  BYTE   smb_pad[];      /* (optional) to pad to word or dword boundary */
  BYTE   smb_data[*];    /* data bytes (* = value of smb_dscnt) */
```

First Response Format :

```
  BYTE    smb_wct;          /* value = 10 */
 WORD     smb_tprcnt;       /* value = 10 */
 WORD     smb_tdrcnt;       /* value = total length of return data buffer  */
 WORD     smb_rsvd;         /* reserved */
 WORD     smb_prcnt;        /* parameter bytes returned in this buffer */
 WORD     smb_proff;        /* offset (from start of SMB hdr) to param bytes */
 WORD     smb_prdisp;       /* value = 0 byte displacement for param bytes */
 WORD     smb_drcnt;        /* data bytes returned in this buffer */
 WORD     smb_droff;        /* offset (from start of SMB hdr) to data bytes */
 WORD     smb_drdisp;       /* byte displacement for these data bytes */
  BYTE    smb_suwcnt;       /* value = 0 no set up return words */
  BYTE    smb_rsvd1;        /* reserved (pad above to word) */
 WORD     smb_bcc;          /* total bytes (including pad bytes) following */
  BYTE    smb_pad[];        /* (optional) to pad to word or dword boundary */
  BYTE    smb_param[*];     /* The parmater block for the
                             * TRANSACT2_FINDFIRST function response is
                             * the find first specific return
                             * information in the following format. */
           WORD             findfirst_dir_handle;    /* Directory search handle */
           WORD             findfirst_searchcount;   /* Number of matching
                                                      * entries found */
           WORD             findfirst_eos;           /* end of search indicator. */
           WORD             findfirst_offerror;      /* error offset if EA error */
           WORD             findfirst_lastname;      /* 0 - server does not require
                                                      *     findnext_FileName[] in order
                                                      *     to continue search.
                                                      * else
                                                      *     offset from start of returned
                                                      *     data to filename of last
                                                      *     found entry returned.
                                                      */
  BYTE    smb_pad1[];       /* (optional) to pad to word or dword boundary */
  BYTE    smb_data[*];      /* return data bytes (* = value of smb_dscnt) */
                            /* The data block contains the level dependent
                             * information about the matches found in the search.
                             *
                             * If bit 2 in the findfirst_flags is set, each
                             * returned file descriptor block will be preceeded
                             * by a four byte resume key.
                             */
```

Subsequent Response Format :

```
BYTE   smb_wct;       /* value = 10 */
WORD   smb_tprcnt;    /* value = 8 */
WORD   smb_tdrcnt;    /* value = total length of return data buffer  */
WORD   smb_rsvd;      /* reserved */
WORD   smb_prcnt;     /* value = 0 */
WORD   smb_proff;     /* value = 0 */
WORD   smb_prdisp;    /* value = 0 */
WORD   smb_drcnt;     /* data bytes returned in this buffer */
WORD   smb_droff;     /* offset (from start of SMB hdr) to data bytes */
WORD   smb_drdisp;    /* byte displacement for these data bytes */
BYTE   smb_suwcnt;    /* value = 0 no set up return words */
BYTE   smb_rsvd1;     /* reserved (pad above to word) */
WORD   smb_bcc;       /* total bytes (including pad bytes) following */
BYTE   smb_pad1[];    /* (optional) to pad to word or dword boundary */
BYTE   smb_data[*];   /* return data bytes (* = value of smb_dscnt) */
                      /* The data block contains the level dependent
                       * information about the matches found in the search.
                       *
                       * If bit 2 in the findfirst_flags is set, each
                       * returned file descriptor block will be preceeded
                       * by a four byte resume key.
                       */
```

6.0.1.1.3. TRANSACT2_FINDNEXT

The function code TRANSACT2_FINDNEXT in smb_setup[0] in the primary TRANSACT2 request identifies a request to continue a file search started by a TRANSACT_FINDFIRST search.

Primary Request Format:

```
  BYTE    smb_wct;              /* value = 15 */
  WORD    smb_tpscnt;           /* total param bytes being sent */
  WORD    smb_tdscnt;           /* total number of data bytes being sent */
  WORD    smb_mprcnt;           /* value = maximum return parameter length */
  WORD    smb_mdrcnt;           /* value = maximum return data length */
  BYTE    smb_msrcnt;           /* value = 0. No setup words to return */
  BYTE    smb_rsvd;             /* reserved (pad above to word) */
  WORD    smb_flags;            /* additional information:
                                    bit 0 - 0
                                    bit 1 - 0 */
 DWORD    smb_timeout;          /* value = 0. Not used for find next */
  WORD    smb_rsvd1;            /* reserved */
  WORD    smb_pscnt;            /* value = tpscnt, parms must be in primary request */
  WORD    smb_psoff;            /* offset (from start of SMB hdr to parameter bytes */
  WORD    smb_dscnt;            /* number of data bytes being sent this buffer */
  WORD    smb_dsoff;            /* offset (from start of SMB hdr) to data bytes */
  BYTE    smb_suwcnt;           /* value = 1 */
  BYTE    smb_rsvd2;            /* reserved (pad above to word) */
  WORD    smb_setup1;           /* value = 2  :- TRANSACT2_FINDNEXT */
  WORD    smb_bcc;              /* total bytes (including pad bytes) following */
  BYTE    smb_pad[];            /* (optional) to pad to word or dword boundary */
  BYTE    smb_param[*];         /* The parmater block for the TRANSACT2_FINDNEXT
                                 * function is the find next specific information
                                 * in the following format. */
           WORD                 findnext_DirHandle;     /* Directory search handle */
           WORD                 findnext_SearchCount;   /* Number of entries to find */
           WORD                 findnext_FileInfoLevel; /* Search level */
          DWORD                 findnext_ResumeKey;     /* Server reserved resume key */
           WORD                 findnext_flags;         /* find flags */
                                                 /* Bit 0: set - close search after
                                                  *              this request.
                                                  * Bit 1: set - close search if end
                                                  *              of search reached.
                                                  * Bit 2: set - Requester requires
                                                  *              resume key for each
                                                  *              entry found.
                                                  * Bit 3: set - Continue search from
                                                  *              last entry returned.
                                                  *        clr - Rewind search. */
  BYTE    findnext_FileName[];  /* Name of file to resume search from */
  BYTE    smb_pad1[];           /* (optional) to pad to word or dword boundary */
  BYTE    smb_data[*];          /* Additional FileInfoLevel dependent match
                                 * information. For a search requiring extended
                                 * attribute matching the data buffer contains
                                 * the FEAList data for the seach.
                                 */
```

Secondary Request Format (more data - may be zero or more of these):

```
 BYTE    smb_wct;        /* value = 9 */
 WORD    smb_tpscnt;     /* total parmeter bytes sent */
 WORD    smb_tdscnt;     /* total number of data bytes being sent */
 WORD    smb_pscnt;      /* value = 0. All params in primary request */
 WORD    smb_psoff;      /* value = 0. No parameters in secondary request. */
 WORD    smb_psdisp;     /* value = 0. No parameters in secondary request. */
 WORD    smb_dscnt;      /* number of data bytes being sent this buffer */
 WORD    smb_dsoff;      /* offset (from start of SMB hdr) to data bytes */
 WORD    smb_dsdisp;     /* byte displacement for these data bytes */
 WORD    smb_fid;        /* search handle returned from find first */
 WORD    smb_bcc;        /* total bytes (including pad bytes) following */
 BYTE    smb_pad[];      /* (optional) to pad to word or dword boundary */
 BYTE    smb_data[*];    /* data bytes (* = value of smb_dscnt) */
```

First Response Format :

```
  BYTE    smb_wct;          /* value = 10 */
 WORD     smb_tprcnt;       /* value = 6 */
 WORD     smb_tdrcnt;       /* value = total length of return data buffer  */
 WORD     smb_rsvd;         /* reserved */
 WORD     smb_prcnt;        /* parameter bytes returned in this buffer */
 WORD     smb_proff;        /* offset (from start of SMB hdr) to param bytes */
 WORD     smb_prdisp;       /* value = 0 byte displacement for param bytes */
 WORD     smb_drcnt;        /* data bytes returned in this buffer */
 WORD     smb_droff;        /* offset (from start of SMB hdr) to data bytes */
 WORD     smb_drdisp;       /* byte displacement for these data bytes */
  BYTE    smb_suwcnt;       /* value = 0 no set up return words */
  BYTE    smb_rsvd1;        /* reserved (pad above to word) */
 WORD     smb_bcc;          /* total bytes (including pad bytes) following */
  BYTE    smb_pad[];        /* (optional) to pad to word or dword boundary */
  BYTE    smb_param[*];     /* The parmater block for the TRANSACT2_FINDNEXT
                             * function response is the find next specific return
                             * information in the following format. */
          WORD              findnext_searchcount;   /* Number of matching
                                                     * entries found */
          WORD              findnext_eos;           /* end of search indicator. */
          WORD              findnext_offerror;      /* error offset if EA error */
          WORD              findfirst_lastname;     /* 0 - server does not require
                                                     *     findnext_FileName[] in order
                                                     *     to continue search.
                                                     * else
                                                     *     offset from start of returned
                                                     *     data to filename of last
                                                     *     found entry returned.
                                                     */
  BYTE    smb_pad1[];       /* (optional) to pad to word or dword boundary */
  BYTE    smb_data[*];      /* return data bytes (* = value of smb_dscnt) */
                            /* The data block contains the level dependent
                             * information about the matches found in the search.
                             *
                             * If bit 2 in the findfirst_flags is set, each
                             * returned file descriptor block will be preceeded
                             * by a four byte resume key.
                             */
```

Subsequent Response Format :

```
 BYTE    smb_wct;       /* value = 10 */
WORD     smb_tprcnt;    /* value = 6 */
WORD     smb_tdrcnt;    /* value = total length of return data buffer  */
WORD     smb_rsvd;      /* reserved */
WORD     smb_prcnt;     /* value = 0 */
WORD     smb_proff;     /* value = 0 */
WORD     smb_prdisp;    /* value = 0 */
WORD     smb_drcnt;     /* data bytes returned in this buffer */
WORD     smb_droff;     /* offset (from start of SMB hdr) to data bytes */
WORD     smb_drdisp;    /* byte displacement for these data bytes */
 BYTE    smb_suwcnt;    /* value = 0 no set up return words */
 BYTE    smb_rsvd1;     /* reserved (pad above to word) */
WORD     smb_bcc;       /* total bytes (including pad bytes) following */
 BYTE    smb_pad1[];    /* (optional) to pad to word or dword boundary */
 BYTE    smb_data[*];   /* return data bytes (* = value of smb_dscnt) */
                        /* The data block contains the level dependent
                         * information about the matches found in the search.
                         *
                         * If bit 2 in the findfirst_flags is set, each
                         * returned file descriptor block will be preceeded
                         * by a four byte resume key.
                         */
```

6.0.1.1.4. TRANSACT2_QFSINFO

The function code TRANSACT2_QFSINFO in smb_setup[0] in the primary TRANSACT2 requests identifies a request to query information about a file system.

Primary Request Format:

```
  BYTE    smb_wct;          /* value = 15 */
  WORD    smb_tpscnt;       /* value = 2, total parameter bytes being sent */
  WORD    smb_tdscnt;       /* total number of data bytes being sent */
  WORD    smb_mprcnt;       /* value = maximum return parameter length */
  WORD    smb_mdrcnt;       /* maximum data length to return */
  BYTE    smb_msrcnt;       /* value = 0. No setup words to return */
  BYTE    smb_rsvd;         /* reserved (pad above to word) */
  WORD    smb_flags;        /* additional information:
                               bit 0 - 0
                               bit 1 - 0   */
 DWORD    smb_timeout;      /* value = 0. Not used for qfsinfo */
  WORD    smb_rsvd1;        /* reserved */
  WORD    smb_pscnt;        /* value = 2, params are in primary request */
  WORD    smb_psoff;        /* offset (from start of SMB Hdr to parameter bytes */
  WORD    smb_dscnt;        /* value = 0, no data sent with qfsinfo */
  WORD    smb_dsoff;        /* value = 0, no data sent with qfsinfo */
  BYTE    smb_suwcnt;       /* value = 1 */
  BYTE    smb_rsvd2;        /* reserved (pad above to word) */
  WORD    smb_setup1;       /* value = 3  :- TRANSACT2_QFSINFO */
  WORD    smb_bcc;          /* total bytes (including pad bytes) following */
  BYTE    smb_pad[];        /* (optional) to pad to word or dword boundary */
  BYTE    smb_param[*];     /* The parmater block for the
                             * TRANSACT2_QFSINFO function is
                             * the qfsinfo specific information
                             * in the following format. */
          WORD              qfsinfo_FSInfoLevel;  /* Level of information required */
```

Response Format (One or more of these) :

```
  BYTE    smb_wct;        /* value = 10 */
  WORD    smb_tprcnt;     /* value = 0 */
  WORD    smb_tdrcnt;     /* value = total length of return data buffer  */
  WORD    smb_rsvd;       /* reserved */
  WORD    smb_prcnt;      /* value = 0, no return param bytes for QFSINFO */
  WORD    smb_proff;      /* offset (from start of SMB hdr) to param bytes */
  WORD    smb_prdisp;     /* value = 0 byte displacement for param bytes */
  WORD    smb_drcnt;      /* data bytes returned in this buffer */
  WORD    smb_droff;      /* offset (from start of SMB hdr) to data bytes */
  WORD    smb_drdisp;     /* byte displacement for these data bytes */
  BYTE    smb_suwcnt;     /* value = 0 no set up return words */
  BYTE    smb_rsvd1;      /* reserved (pad above to word) */
  WORD    smb_bcc;        /* total bytes (including pad bytes) following */
  BYTE    smb_pad1[];     /* (optional) to pad to word or dword boundary */
  BYTE    smb_data[*];    /* return data bytes (* = value of smb_dscnt) */
                          /* The data block contains the level dependent
                           * information about the file system.
                           */
```

6.0.1.1.5. TRANSACT2_SETFSINFO

The function code TRANSACT2_SETFSINFO in smb_setup[0] in the primary TRANSACT2 requests identifies a request to set information for a file system device.

Primary Request Format:

```
  BYTE    smb_wct;        /* value = 15 */
  WORD    smb_tpscnt;     /* value = 2,total number of param bytes being sent */
  WORD    smb_tdscnt;     /* total number of data bytes being sent */
  WORD    smb_mprcnt;     /* value = maximum return parameter length */
  WORD    smb_mdrcnt;     /* value = 0. No data returned */
  BYTE    smb_msrcnt;     /* value = 0. No setup words to return */
  BYTE    smb_rsvd;       /* reserved (pad above to word) */
  WORD    smb_flags;      /* additional information:
                              bit 0 - 0
                              bit 1 - 0 */
 DWORD    smb_timeout;    /* value = 0. Not used for setfsinfo */
  WORD    smb_rsvd1;      /* reserved */
  WORD    smb_pscnt;      /* value = 4, all params are in primary request */
  WORD    smb_psoff;      /* offset (from start of SMB Hdr to parameter bytes */
  WORD    smb_dscnt;      /* number of data bytes being sent this buffer */
  WORD    smb_dsoff;      /* offset (from start of SMB hdr) to data bytes */
  BYTE    smb_suwcnt;     /* value = 1 */
  BYTE    smb_rsvd2;      /* reserved (pad above to word) */
  WORD    smb_setup1;     /* value = 4  :- TRANSACT2_SETFSINFO */
  WORD    smb_bcc;        /* total bytes (including pad bytes) following */
  BYTE    smb_pad[];      /* (optional) to pad to word or dword boundary */
  BYTE    smb_param[*];   /* The parmater block for the
                           * TRANSACT2_SETFSINFO function is
                           * the setfsinfo specific information
                           * in the following format. */
          WORD            setfsinfo_FSInfoLevel;  /* Level of information
                                                   * provided */
  BYTE    smb_pad1[];     /* (optional) to pad to word or dword boundary */
  BYTE    smb_data[*];    /* Level dependent file system information */
```

Secondary Request Format (more data - may be zero or more of these):

```
  BYTE    smb_wct;        /* value = 9 */
  WORD    smb_tpscnt;     /* totalnumber of parameter bytes being sent */
  WORD    smb_tdscnt;     /* total number of data bytes being sent */
  WORD    smb_pscnt;      /* value = 0. All params in primary request */
  WORD    smb_psoff;      /* value = 0. No parameters in secondary request. */
  WORD    smb_psdisp;     /* value = 0. No parameters in secondary request. */
  WORD    smb_dscnt;      /* number of data bytes being sent this buffer */
  WORD    smb_dsoff;      /* offset (from start of SMB hdr) to data bytes */
  WORD    smb_dsdisp;     /* byte displacement for these data bytes */
  WORD    smb_fid;        /* value = 0xffff, no handle on request */
  WORD    smb_bcc;        /* total bytes (including pad bytes) following */
  BYTE    smb_pad[];      /* (optional) to pad to word or dword boundary */
  BYTE    smb_data[*];    /* data bytes (* = value of smb_dscnt) */
```

Response Format (one only):

```
 BYTE   smb_wct;      /* value = 10 */
WORD    smb_tprcnt;   /* value = 0 */
WORD    smb_tdrcnt;   /* value = 0 no data bytes  */
WORD    smb_rsvd;     /* reserved */
WORD    smb_prcnt;    /* value = 0 no return parameters for setfsinfo */
WORD    smb_proff;    /* offset (from start of SMB hdr) to param bytes */
WORD    smb_prdisp;   /* value = 0 byte displacement for param bytes */
WORD    smb_drcnt;    /* value = 0 no data bytes */
WORD    smb_droff;    /* value = 0 no data bytes */
WORD    smb_drdisp;   /* value = 0 no data bytes */
 BYTE   smb_suwcnt;   /* value = 0 no set up return words */
 BYTE   smb_rsvd1;    /* reserved (pad above to word) */
WORD    smb_bcc;      /* value = 0 */
```

6.0.1.1.6. TRANSACT2_QPATHINFO

The function code TRANSACT2_QPATHINFO in smb_setup[0] in the primary TRANSACT2 requests identifies a request to query information about specific file or subdirectory.

Primary Request Format:

```
  BYTE    smb_wct;        /* value = 15 */
  WORD    smb_tpscnt;     /* value = total number of param bytes being sent */
  WORD    smb_tdscnt;     /* total number of data bytes being sent */
  WORD    smb_mprcnt;     /* value = maximum return parameter length */
  WORD    smb_mdrcnt;     /* maximum data length to return */
  BYTE    smb_msrcnt;     /* value = 0. No setup words to return */
  BYTE    smb_rsvd;       /* reserved (pad above to word) */
  WORD    smb_flags;      /* additional information:
                              bit 0 - 0
                              bit 1 - 0 */
 DWORD    smb_timeout;    /* value = 0. Not used for qpathinfo */
  WORD    smb_rsvd1;      /* reserved */
  WORD    smb_pscnt;      /* value = tpscnt, all params are in primary request */
  WORD    smb_psoff;      /* offset (from start of SMB hdr) to parameter bytes */
  WORD    smb_dscnt;      /* number of data bytes being sent this buffer */
  WORD    smb_dsoff;      /* offset (from start of SMB hdr) to data bytes */
  BYTE    smb_suwcnt;     /* value = 1 */
  BYTE    smb_rsvd2;      /* reserved (pad above to word) */
  WORD    smb_setup1;     /* value = 5  :- TRANSACT2_QPATHINFO */
  WORD    smb_bcc;        /* total bytes (including pad bytes) following */
  BYTE    smb_pad[];      /* (optional) to pad to word or dword boundary */
  BYTE    smb_param[*];   /* The parmater block for the
                           * TRANSACT2_QPATHINFO function is the
                           * qpathinfo specific information
                           * in the following format. */
           WORD           qpathinfo_PathInfoLevel; /* Info level required. */
          DWORD           qpathinfo_rsvd;          /* Reserved.
                                                    * Must be zero. */
          BYTE            qpathinfo_PathName[];    /* File/directory name. */
  BYTE    smb_pad1[];     /* (optional) to pad to word or dword boundary */
  BYTE    smb_data[*];    /* Additional FileInfoLevel dependent information */
```

Secondary Request Format (more data - may be zero or more of these):

```
  BYTE    smb_wct;       /* value = 9 */
  WORD    smb_tpscnt;    /* totalnumber of parameter bytes being sent */
  WORD    smb_tdscnt;    /* total number of data bytes being sent */
  WORD    smb_pscnt;     /* value = 0. All params in primary request */
  WORD    smb_psoff;     /* value = 0. No parameters in secondary request. */
  WORD    smb_psdisp;    /* value = 0. No parameters in secondary request. */
  WORD    smb_dscnt;     /* number of data bytes being sent this buffer */
  WORD    smb_dsoff;     /* offset (from start of SMB hdr) to data bytes */
  WORD    smb_dsdisp;    /* byte displacement for these data bytes */
  WORD    smb_fid;       /* value = 0xffff, no handle on request */
  WORD    smb_bcc;       /* total bytes (including pad bytes) following */
  BYTE    smb_pad[];     /* (optional) to pad to word or dword boundary */
  BYTE    smb_data[*];   /* data bytes (* = value of smb_dscnt) */
```

First Response Format :

```
  BYTE    smb_wct;          /* value = 10 */
 WORD     smb_tprcnt;       /* value = 2 */
 WORD     smb_tdrcnt;       /* value = total length of return data buffer  */
 WORD     smb_rsvd;         /* reserved */
 WORD     smb_prcnt;        /* value = 2 param bytes returned for QFSINFO */
 WORD     smb_proff;        /* offset (from start of SMB hdr) to param bytes */
 WORD     smb_prdisp;       /* value = 0 byte displacement for param bytes */
 WORD     smb_drcnt;        /* data bytes returned in this buffer */
 WORD     smb_droff;        /* offset (from start of SMB hdr) to data bytes */
 WORD     smb_drdisp;       /* byte displacement for these data bytes */
  BYTE    smb_suwcnt;       /* value = 0 no set up return words */
  BYTE    smb_rsvd1;        /* reserved (pad above to word) */
 WORD     smb_bcc;          /* total bytes (including pad bytes) following */
  BYTE    smb_pad[];        /* (optional) to pad to word or dword boundary */
  BYTE    smb_param[*];     /* The parmater block for the
                             * TRANSACT2_QPATHINFO response is
                             * the qpathinfo specific return
                             * information in the following format. */
           WORD             qpathinfo_offerror;     /* error offset if EA error */
  BYTE    smb_pad1[];       /* (optional) to pad to word or dword boundary */
  BYTE    smb_data[*];      /* return data bytes (* = value of smb_dscnt) */
                            /* The data block contains the requested level
                             * dependent information about the path.
                             */
```

Subsequent Response Format :

```
  BYTE    smb_wct;        /* value = 10 */
 WORD     smb_tprcnt;     /* value = 2 */
 WORD     smb_tdrcnt;     /* value = total length of return data buffer  */
 WORD     smb_rsvd;       /* reserved */
 WORD     smb_prcnt;      /* value = 0 */
 WORD     smb_proff;      /* value = 0 */
 WORD     smb_prdisp;     /* value = 0 */
 WORD     smb_drcnt;      /* data bytes returned in this buffer */
 WORD     smb_droff;      /* offset (from start of SMB hdr) to data bytes */
 WORD     smb_drdisp;     /* byte displacement for these data bytes */
  BYTE    smb_suwcnt;     /* value = 0 no set up return words */
  BYTE    smb_rsvd1;      /* reserved (pad above to word) */
 WORD     smb_bcc;        /* total bytes (including pad bytes) following */
  BYTE    smb_pad1[];     /* (optional) to pad to word or dword boundary */
  BYTE    smb_data[*];    /* return data bytes (* = value of smb_dscnt) */
                          /* The data block contains the requested level
                           * dependent information about the path.
                           */
```

6.0.1.1.7. TRANSACT2_SETPATHINFO

The function code TRANSACT2_SETPATHINFO in smb_setup[0] in the primary TRANSACT2 requests identifies a request to set information for a file or directory.

Primary Request Format:

```
  BYTE    smb_wct;          /* value = 15 */
  WORD    smb_tpscnt;       /* value = total number of param bytes being sent */
  WORD    smb_tdscnt;       /* total number of data bytes being sent */
  WORD    smb_mprcnt;       /* value = maximum return parameter length */
  WORD    smb_mdrcnt;       /* value = 0. No data returned */
  BYTE    smb_msrcnt;       /* value = 0. No setup words to return */
  BYTE    smb_rsvd;         /* reserved (pad above to word) */
  WORD    smb_flags;        /* additional information:
                                bit 0 - 0
                                bit 1 - 0  */
 DWORD    smb_timeout;      /* value = 0. Not used for setpathinfo */
  WORD    smb_rsvd1;        /* reserved */
  WORD    smb_pscnt;        /* value = tpscnt, params are in primary request */
  WORD    smb_psoff;        /* offset (from start of SMB Hdr to param bytes */
  WORD    smb_dscnt;        /* number of data bytes being sent this buffer */
  WORD    smb_dsoff;        /* offset (from start of SMB hdr) to data bytes */
  BYTE    smb_suwcnt;       /* value = 1 */
  BYTE    smb_rsvd2;        /* reserved (pad above to word) */
  WORD    smb_setup1;       /* value = 6  :- TRANSACT2_SETPATHINFO */
  WORD    smb_bcc;          /* total bytes (including pad bytes) following */
  BYTE    smb_pad[];        /* (optional) to pad to word or dword boundary */
  BYTE    smb_param[*];     /* The parmater block for the
                             * TRANSACT2_SETPATHINFO function is
                             * the setpathinfo specific information
                             * in the following format. */
           WORD             setpathinfo_PathInfoLevel;  /* Info level supplied. */
          DWORD             setpathinfo_rsvd;           /* Reserved.
                                                         * Must be zero. */
           BYTE             setpathinfo_pathname[];     /* path name to set
                                                         * information on */
  BYTE    smb_pad1[];       /* (optional) to pad to word or dword boundary */
  BYTE    smb_data[*];      /* Additional FileInfoLevel dependent information. */
```

Secondary Request Format (more data - may be zero or more of these):

```
  BYTE    smb_wct;        /* value = 9 */
  WORD    smb_tpscnt;     /* totalnumber of parameter bytes being sent */
  WORD    smb_tdscnt;     /* total number of data bytes being sent */
  WORD    smb_pscnt;      /* value = 0. All params in primary request */
  WORD    smb_psoff;      /* value = 0. No parameters in secondary request. */
  WORD    smb_psdisp;     /* value = 0. No parameters in secondary request. */
  WORD    smb_dscnt;      /* number of data bytes being sent this buffer */
  WORD    smb_dsoff;      /* offset (from start of SMB hdr) to data bytes */
  WORD    smb_dsdisp;     /* byte displacement for these data bytes */
  WORD    smb_fid;        /* value = 0xffff, no handle on request */
  WORD    smb_bcc;        /* total bytes (including pad bytes) following */
  BYTE    smb_pad[];      /* (optional) to pad to word or dword boundary */
  BYTE    smb_data[*];    /* data bytes (* = value of smb_dscnt) */
```

Response Format (one only):

```
 BYTE   smb_wct;          /* value = 10 */
WORD    smb_tprcnt;       /* value = 2 */
WORD    smb_tdrcnt;       /* value = 0 no data bytes  */
WORD    smb_rsvd;         /* reserved */
WORD    smb_prcnt;        /* value = 2 parameter bytes being returned */
WORD    smb_proff;        /* offset (from start of SMB hdr) to param bytes */
WORD    smb_prdisp;       /* value = 0 byte displacement for param bytes */
WORD    smb_drcnt;        /* value = 0 no data bytes */
WORD    smb_droff;        /* value = 0 no data bytes */
WORD    smb_drdisp;       /* value = 0 no data bytes */
 BYTE   smb_suwcnt;       /* value = 0 no set up return words */
 BYTE   smb_rsvd1;        /* reserved (pad above to word) */
WORD    smb_bcc;          /* total bytes (including pad bytes) following */
 BYTE   smb_pad[];        /* (optional) to pad to word or dword boundary */
 BYTE   smb_param[*];     /* The parmater block for the
                           * TRANSACT2_SETPATHINFO function
                           * response is the setpathinfo
                           * specific return information in
                           * the following format. */
        WORD              setpathinfo_offerror;  /* offset into FEAList data
                                                  * of first error which
                                                  * occured while setting
                                                  * the extended attributes. */
```

6.0.1.1.8. TRANSACT2_QFILEINFO

The function code TRANSACT2_QFILEINFO in smb_setup[0] in the primary TRANSACT2 requests identifies a request to query information about specific file.

Primary Request Format:

```
  BYTE    smb_wct;         /* value = 15 */
  WORD    smb_tpscnt;      /* value = 4,total number of param bytes being sent */
  WORD    smb_tdscnt;      /* total number of data bytes being sent */
  WORD    smb_mprcnt;      /* value = maximum return parameter length */
  WORD    smb_mdrcnt;      /* maximum data length to return */
  BYTE    smb_msrcnt;      /* value = 0. No setup words to return */
  BYTE    smb_rsvd;        /* reserved (pad above to word) */
  WORD    smb_flags;       /* additional information:
                               bit 0 - 0
                               bit 1 - 0  */
 DWORD    smb_timeout;     /* value = 0. Not used for qfileinfo */
  WORD    smb_rsvd1;       /* reserved */
  WORD    smb_pscnt;       /* value = 4, all params are in primary request */
  WORD    smb_psoff;       /* offset (from start of SMB hdr) to param bytes */
  WORD    smb_dscnt;       /* number of data bytes being sent this buffer */
  WORD    smb_dsoff;       /* offset (from start of SMB hdr) to data bytes */
  BYTE    smb_suwcnt;      /* value = 1 */
  BYTE    smb_rsvd2;       /* reserved (pad above to word) */
  WORD    smb_setup1;      /* value = 7  :- TRANSACT2_QFILEINFO */
  WORD    smb_bcc;         /* total bytes (including pad bytes) following */
  BYTE    smb_pad[];       /* (optional) to pad to word or dword boundary */
  BYTE    smb_param[*];    /* The parmater block for the
                            * TRANSACT2_QFILEINFO function
                            * is the qfileinfo specific information
                            * in the following format. */
          WORD             qfileinfo_FileHandle;     /* File handle. */
          WORD             qfileinfo_FileInfoLevel; /* Info level required. */
  BYTE    smb_pad1[];      /* (optional) to pad to word or dword boundary */
  BYTE    smb_data[*];     /* Additional FileInfoLevel dependent information. */
```

Secondary Request Format (more data - may be zero or more of these):

```
  BYTE    smb_wct;        /* value = 9 */
  WORD    smb_tpscnt;     /* totalnumber of parameter bytes being sent */
  WORD    smb_tdscnt;     /* total number of data bytes being sent */
  WORD    smb_pscnt;      /* value = 0. All params in primary request */
  WORD    smb_psoff;      /* value = 0. No parameters in secondary request. */
  WORD    smb_psdisp;     /* value = 0. No parameters in secondary request. */
  WORD    smb_dscnt;      /* number of data bytes being sent this buffer */
  WORD    smb_dsoff;      /* offset (from start of SMB hdr) to data bytes */
  WORD    smb_dsdisp;     /* byte displacement for these data bytes */
  WORD    smb_fid;        /* file handle */
  WORD    smb_bcc;        /* total bytes (including pad bytes) following */
  BYTE    smb_pad[];      /* (optional) to pad to word or dword boundary */
  BYTE    smb_data[*];    /* data bytes (* = value of smb_dscnt) */
```

First Response Format :

```
  BYTE    smb_wct;          /* value = 10 */
 WORD     smb_tprcnt;       /* value = 2 */
 WORD     smb_tdrcnt;       /* value = total length of return data buffer  */
 WORD     smb_rsvd;         /* reserved */
 WORD     smb_prcnt;        /* value = 2 no param bytes returned for qfileinfo */
 WORD     smb_proff;        /* offset (from start of SMB hdr) to param bytes */
 WORD     smb_prdisp;       /* value = 0 byte displacement for param bytes */
 WORD     smb_drcnt;        /* data bytes returned in this buffer */
 WORD     smb_droff;        /* offset (from start of SMB hdr) to data bytes */
 WORD     smb_drdisp;       /* byte displacement for these data bytes */
  BYTE    smb_suwcnt;       /* value = 0 no set up return words */
  BYTE    smb_rsvd1;        /* reserved (pad above to word) */
 WORD     smb_bcc;          /* total bytes (including pad bytes) following */
  BYTE    smb_pad[];        /* (optional) to pad to word or dword boundary */
  BYTE    smb_param[*];     /* The parmater block for the
                             * TRANSACT2_QFILEINFO response is
                             * the qfileinfo specific return
                             * information in the following format. */
           WORD               qfileinfo_offerror;     /* error offset if EA error */
  BYTE    smb_pad1[];       /* (optional) to pad to word or dword boundary */
  BYTE    smb_data[*];      /* return data bytes (* = value of smb_dscnt) */
                            /* The data block contains the requested level
                             * dependent information about the file. */
```

Subsequent Response Format :

```
  BYTE    smb_wct;          /* value = 10 */
 WORD     smb_tprcnt;       /* value = 2 */
 WORD     smb_tdrcnt;       /* value = total length of return data buffer  */
 WORD     smb_rsvd;         /* reserved */
 WORD     smb_prcnt;        /* value = 0 */
 WORD     smb_proff;        /* value = 0 */
 WORD     smb_prdisp;       /* value = 0 */
 WORD     smb_drcnt;        /* data bytes returned in this buffer */
 WORD     smb_droff;        /* offset (from start of SMB hdr) to data bytes */
 WORD     smb_drdisp;       /* byte displacement for these data bytes */
  BYTE    smb_suwcnt;       /* value = 0 no set up return words */
  BYTE    smb_rsvd1;        /* reserved (pad above to word) */
 WORD     smb_bcc;          /* total bytes (including pad bytes) following */
  BYTE    smb_pad1[];       /* (optional) to pad to word or dword boundary */
  BYTE    smb_data[*];      /* return data bytes (* = value of smb_dscnt) */
                            /* The data block contains the requested level
                             * dependent information about the file. */
```

6.0.1.1.9. TRANSACT2_SETFILEINFO

The function code TRANSACT2_SETFILEINFO in smb_setup[0] in the primary TRANSACT2 requests identifies a request to set information for a specific file.

Primary Request Format:

```
  BYTE    smb_wct;         /* value = 15 */
  WORD    smb_tpscnt;      /* value = 6, total param bytes being sent */
  WORD    smb_tdscnt;      /* total number of data bytes being sent */
  WORD    smb_mprcnt;      /* value = maximum return parameter length */
  WORD    smb_mdrcnt;      /* value = 0. No data returned */
  BYTE    smb_msrcnt;      /* value = 0. No setup words to return */
  BYTE    smb_rsvd;        /* reserved (pad above to word) */
  WORD    smb_flags;       /* additional information:
                               bit 0 - 0
                               bit 1 - 0  */
 DWORD    smb_timeout;     /* value = 0. Not used for setfileinfo */
  WORD    smb_rsvd1;       /* reserved */
  WORD    smb_pscnt;       /* value = 6, parms must be in primary request */
  WORD    smb_psoff;       /* offset (from start of SMB Hdr to parameter bytes */
  WORD    smb_dscnt;       /* number of data bytes being sent this buffer */
  WORD    smb_dsoff;       /* offset (from start of SMB hdr) to data bytes */
  BYTE    smb_suwcnt;      /* value = 1 */
  BYTE    smb_rsvd2;       /* reserved (pad above to word) */
  WORD    smb_setup1;      /* value = 8  :- TRANSACT2_SETFILEINFO */
  WORD    smb_bcc;         /* total bytes (including pad bytes) following */
  BYTE    smb_pad[];       /* (optional) to pad to word or dword boundary */
  BYTE    smb_param[*];    /* The parmater block for the
                            * TRANSACT2_SETFILEINFO function is
                            * the setfileinfo specific information
                            * in the following format. */
          WORD             setfileinfo_FileHandle;     /* File handle. */
          WORD             setfileinfo_FileInfoLevel; /* Info level supplied. */
          WORD             setfileinfo_IOFlag;        /* Flag
                                                       * 0x0010 - Write through
                                                       * 0x0020 - No cache */
  BYTE    smb_pad1[];      /* (optional) to pad to word or dword boundary */
  BYTE    smb_data[*];     /* Additional FileInfoLevel dependent information.
                           /* For level = 2, smb_data[] contains the FEAList
                            * structure to set for this file. */
```

Secondary Request Format (more data - may be zero or more of these):

```
 BYTE   smb_wct;        /* value = 9 */
 WORD   smb_tpscnt;     /* value = 4 */
 WORD   smb_tdscnt;     /* total number of data bytes being sent */
 WORD   smb_pscnt;      /* value = 0. All params in primary request */
 WORD   smb_psoff;      /* value = 0. No parameters in secondary request. */
 WORD   smb_psdisp;     /* value = 0. No parameters in secondary request. */
 WORD   smb_dscnt;      /* number of data bytes being sent this buffer */
 WORD   smb_dsoff;      /* offset (from start of SMB hdr) to data bytes */
 WORD   smb_dsdisp;     /* byte displacement for these data bytes */
 WORD   smb_fid;        /* file handle */
 WORD   smb_bcc;        /* total bytes (including pad bytes) following */
 BYTE   smb_pad[];      /* (optional) to pad to word or dword boundary */
 BYTE   smb_data[*];    /* data bytes (* = value of smb_dscnt) */
```

Response Format (one only):

```
 BYTE   smb_wct;         /* value = 10 */
 WORD   smb_tprcnt;      /* value = 2 */
 WORD   smb_tdrcnt;      /* value = 0 no data bytes  */
 WORD   smb_rsvd;        /* reserved */
 WORD   smb_prcnt;       /* value = 2 parameter bytes being returned */
 WORD   smb_proff;       /* offset (from start of SMB hdr) to param bytes */
 WORD   smb_prdisp;      /* value = 0, byte displacement for these params */
 WORD   smb_drcnt;       /* value = 0 no data bytes */
 WORD   smb_droff;       /* value = 0 no data bytes */
 WORD   smb_drdisp;      /* value = 0 no data bytes */
 BYTE   smb_suwcnt;      /* value = 0 no set up return words */
 BYTE   smb_rsvd1;       /* reserved (pad above to word) */
 WORD   smb_bcc;         /* total bytes (including pad bytes) following */
 BYTE   smb_pad[];       /* (optional) to pad to word or dword boundary */
 BYTE   smb_param[*];    /* The parmater block for the
                          * TRANSACT2_SETFILEINFO function
                          * response is the setfileinfo specific
                          * return information in the
                          * following format. */
        WORD             setfileinfo_offerror;  /* offset into FEAList
                                                 * data of first error
                                                 * which occured while
                                                 * setting the extended
                                                 * attributes. */
```

6.0.1.1.10. TRANSACT2_FSCTL

The function code TRANSACT2_FSCTL in smb_setup[0] in the primary TRANSACT2 requests identifies a file system control request.

Primary Request Format:

```
  BYTE    smb_wct;        /* value = 14 + value of smb_suwcnt */
  WORD    smb_tpscnt;     /* value = total number of param bytes being sent */
  WORD    smb_tdscnt;     /* total number of data bytes being sent */
  WORD    smb_mprcnt;     /* value = maximum return parameter length */
  WORD    smb_mdrcnt;     /* maximum data length to return */
  BYTE    smb_msrcnt;     /* value = 1. Function return code */
  BYTE    smb_rsvd;       /* reserved (pad above to word) */
  WORD    smb_flags;      /* additional information:
                              bit 0 - 0
                              bit 1 - 0 */
 DWORD    smb_timeout;    /* value = 0. Not used for fsctl */
  WORD    smb_rsvd1;      /* reserved */
  WORD    smb_pscnt;      /* number of param bytes being sent in this buffer */
  WORD    smb_psoff;      /* offset (from start of SMB hdr) to parameter bytes */
  WORD    smb_dscnt;      /* number of data bytes being sent this buffer */
  WORD    smb_dsoff;      /* offset (from start of SMB hdr) to data bytes */
  BYTE    smb_suwcnt;     /* value = number of setup words in this buffer */
  BYTE    smb_rsvd2;      /* reserved (pad above to word) */
  WORD    smb_setup[];    /* The setup word array for the
                           * TRANSACT2_FSINFO function is the
                           * fsctl specific information
                           * in the following format. */
          WORD            9 :- TRANSACT2_FSCTL;   /* TRANS2 command code. */
          WORD            fsctl_FileHandle;       /* File handle. */
          WORD            fsctl_Function code;    /* FsCtl function code */
          WORD            fsctl_RouteMethod;      /* Method for routing. */
          BYTE            fsctl_RouteName[];      /* The route name byte
                                                   * array is zero padded
                                                   * to an even length. */
  WORD    smb_bcc;        /* total bytes (including pad bytes) following */
  BYTE    smb_pad[];      /* (optional) to pad to word or dword boundary */
  BYTE    smb_param[*];   /* File system specific parameter block. */
  BYTE    smb_pad1[];     /* (optional) to pad to word or dword boundary */
  BYTE    smb_data[*];    /* File system specific data block. */
```

Secondary Request Format (more data - may be zero or more of these):

```
 BYTE    smb_wct;          /* value = 9 */
WORD     smb_tpscnt;       /* totalnumber of parameter bytes being sent */
WORD     smb_tdscnt;       /* total number of data bytes being sent */
WORD     smb_pscnt;        /* number of parameter bytes being sent this buffer */
WORD     smb_psoff;        /* offset (from start of SMB hdr) to parameter bytes */
WORD     smb_psdisp;       /* byte displacement for these parameter bytes */
WORD     smb_dscnt;        /* number of data bytes being sent this buffer */
WORD     smb_dsoff;        /* offset (from start of SMB hdr) to data bytes */
WORD     smb_dsdisp;       /* byte displacement for these data bytes */
WORD     smb_fid;          /* file handle */
WORD     smb_bcc;          /* total bytes (including pad bytes) following */
 BYTE    smb_pad[];        /* (optional) to pad to word or dword boundary */
 BYTE    smb_param[*];     /* File system specific parameter block. */
 BYTE    smb_pad1[];       /* (optional) to pad to word or dword boundary */
 BYTE    smb_data[*];      /* data bytes (* = value of smb_dscnt) */
```

Response Format :

```
 BYTE    smb_wct;          /* value = 10 */
WORD     smb_tprcnt;       /* value = total length of return parameter buffer  */
WORD     smb_tdrcnt;       /* value = total length of return data buffer  */
WORD     smb_rsvd;         /* reserved */
WORD     smb_prcnt;        /* parameter bytes returned in this buffer */
WORD     smb_proff;        /* offset (from start of SMB hdr) to parameter bytes */
WORD     smb_prdisp;       /* value = 0 byte displacement for these param bytes */
WORD     smb_drcnt;        /* data bytes returned in this buffer */
WORD     smb_droff;        /* offset (from start of SMB hdr) to data bytes */
WORD     smb_drdisp;       /* byte displacement for these data bytes */
 BYTE    smb_suwcnt;       /* value = 0, no set up return words */
 BYTE    smb_rsvd1;        /* reserved (pad above to word) */
WORD     smb_bcc;          /* total bytes (including pad bytes) following */
 BYTE    smb_pad[];        /* (optional) to pad to word or dword boundary */
 BYTE    smb_param[*];     /* File system specific return parameter block */
 BYTE    smb_pad1[];       /* (optional) to pad to word or dword boundary */
 BYTE    smb_data[*];      /* File system specific return data block. */
```

6.0.1.1.11. TRANSACT2_IOCTL

The function code TRANSACT2_IOCTL in smb_setup[0] in the primary TRANSACT2 requests identifies a device control request.

Primary Request Format:

```
  BYTE    smb_wct;         /* value = 18 */
  WORD    smb_tpscnt;      /* value = total number of param bytes being sent */
  WORD    smb_tdscnt;      /* total number of data bytes being sent */
  WORD    smb_mprcnt;      /* value = maximum return parameter length */
  WORD    smb_mdrcnt;      /* maximum data length to return */
  BYTE    smb_msrcnt;      /* value = 1. Function return code */
  BYTE    smb_rsvd;        /* reserved (pad above to word) */
  WORD    smb_flags;       /* additional information:
                               bit 0 - 0
                               bit 1 - 0 */
 DWORD    smb_timeout;     /* value = 0. Not used for fsctl */
  WORD    smb_rsvd1;       /* reserved */
  WORD    smb_pscnt;       /* number of param bytes being sent in this buffer */
  WORD    smb_psoff;       /* offset (from start of SMB hdr) to parameter bytes */
  WORD    smb_dscnt;       /* number of data bytes being sent this buffer */
  WORD    smb_dsoff;       /* offset (from start of SMB hdr) to data bytes */
  BYTE    smb_suwcnt;      /* value = number of setup words in this buffer */
  BYTE    smb_rsvd2;       /* reserved (pad above to word) */
  WORD    smb_setup[];     /* The setup word array for the
                            * TRANSACT2_IOCTL function is the ioctl
                            * functiom specific information
                            * in the following format. */
          WORD             10 :- TRANSACT2_IOCTL;  /* Function code. */
          WORD             ioctl_DevHandle;        /* Device handle. */
          WORD             ioctl_Category;         /* Device catgory. */
          WORD             ioctl_Function;         /* Device function. */
  WORD    smb_bcc;         /* total bytes (including pad bytes) following */
  BYTE    smb_pad[];       /* (optional) to pad to word or dword boundary */
  BYTE    smb_param[*];    /* Device/function specific parameter block. */
  BYTE    smb_pad1[];      /* (optional) to pad to word or dword boundary */
  BYTE    smb_data[*];     /* Device/function specific data block. */
```

Secondary Request Format (more data - may be zero or more of these):

```
 BYTE    smb_wct;          /* value = 9 */
WORD     smb_tpscnt;       /* totalnumber of parameter bytes being sent */
WORD     smb_tdscnt;       /* total number of data bytes being sent */
WORD     smb_pscnt;        /* number of parameter bytes being sent this buffer */
WORD     smb_psoff;        /* offset (from start of SMB hdr) to parameter bytes */
WORD     smb_psdisp;       /* byte displacement for these parameter bytes */
WORD     smb_dscnt;        /* number of data bytes being sent this buffer */
WORD     smb_dsoff;        /* offset (from start of SMB hdr) to data bytes */
WORD     smb_dsdisp;       /* byte displacement for these data bytes */
WORD     smb_fid;          /* file handle */
WORD     smb_bcc;          /* total bytes (including pad bytes) following */
 BYTE    smb_pad[];        /* (optional) to pad to word or dword boundary */
 BYTE    smb_param[*];     /* Device/function specific parameter block. */
 BYTE    smb_pad1[];       /* (optional) to pad to word or dword boundary */
 BYTE    smb_data[*];      /* data bytes (* = value of smb_dscnt) */
```

Response Format :

```
 BYTE    smb_wct;          /* value = 10 */
WORD     smb_tprcnt;       /* value = total length of return parameter buffer  */
WORD     smb_tdrcnt;       /* value = total length of return data buffer  */
WORD     smb_rsvd;         /* reserved */
WORD     smb_prcnt;        /* parameter bytes returned in this buffer */
WORD     smb_proff;        /* offset (from start of SMB hdr) to parameter bytes */
WORD     smb_prdisp;       /* value = 0 byte displacement for these param bytes */
WORD     smb_drcnt;        /* data bytes returned in this buffer */
WORD     smb_droff;        /* offset (from start of SMB hdr) to data bytes */
WORD     smb_drdisp;       /* byte displacement for these data bytes */
 BYTE    smb_suwcnt;       /* value = 0, no set up return words */
 BYTE    smb_rsvd1;        /* reserved (pad above to word) */
WORD     smb_bcc;          /* total bytes (including pad bytes) following */
 BYTE    smb_pad[];        /* (optional) to pad to word or dword boundary */
 BYTE    smb_param[*];     /* Device/function specific return parameter block */
 BYTE    smb_pad1[];       /* (optional) to pad to word or dword boundary */
 BYTE    smb_data[*];      /* Device/function specific return data block. */
```

6.0.1.1.12. TRANSACT2_FINDNOTIFYFIRST

The function code TRANSACT2_FINDNOTIFYFIRST in smb_setup[0] in the primary TRANSACT2 request identifies a request to commence monitoring changes to a specific file or directory.

Primary Request Format:

```
  BYTE    smb_wct;          /* value = 15 */
  WORD    smb_tpscnt;       /* value = total number of param bytes being sent */
  WORD    smb_tdscnt;       /* total size of extended attribute list */
  WORD    smb_mprcnt;       /* value = maximum return parameter length */
  WORD    smb_mdrcnt;       /* value = maximum return data length */
  BYTE    smb_msrcnt;       /* value = 0. No setup words to return */
  BYTE    smb_rsvd;         /* reserved (pad above to word) */
  WORD    smb_flags;        /* additional information:
                               bit 0 - 0
                               bit 1 - 0
 DWORD    smb_timeout;      /* Specifies duration to wait for changes */
  WORD    smb_rsvd1;        /* reserved */
  WORD    smb_pscnt;        /* value = tpscnt, parms must be in primary request */
  WORD    smb_psoff;        /* offset (from start of SMB hdr to parameter bytes */
  WORD    smb_dscnt;        /* number of data bytes being sent this buffer */
  WORD    smb_dsoff;        /* offset (from start of SMB hdr) to data bytes */
  BYTE    smb_suwcnt;       /* value = 1 */
  BYTE    smb_rsvd2;        /* reserved (pad above to word) */
  WORD    smb_setup1;       /* value = 11  :- TRANSACT2_FINDNOTIFYFIRST */
  WORD    smb_bcc;          /* total bytes (including pad bytes) following */
  BYTE    smb_pad[];        /* (optional) to pad to word or dword boundary */
  BYTE    smb_param[*];     /* The parmater block for the
                              TRANSACT2_FINDNOTIFYFIRST function is the find
                              first specific information in the
                              following format. */
           WORD             findnfirst_Attribute;     /* Search attribute */
           WORD             findnfirst_ChangeCount;   /* Number of changes
                                                       * to wait for */
           WORD             findnfirst_Level;         /* Info level required */
          DWORD             findfirst_rsvd;           /* Reserved (must be zero) */
           BYTE             findnfirst_PathSpec[];
  BYTE    smb_pad1[];       /* (optional) to pad to word or dword boundary */
  BYTE    smb_data[*];      /* Additional level dependent match data */
```

Secondary Request Format (more data - may be zero or more of these):

```
  BYTE    smb_wct;        /* value = 9 */
 WORD     smb_tpscnt;     /* totalnumber of parameter bytes being sent */
 WORD     smb_tdscnt;     /* total number of data bytes being sent */
 WORD     smb_pscnt;      /* value = 0. All params in primary request */
 WORD     smb_psoff;      /* value = 0. No parameters in secondary request. */
 WORD     smb_psdisp;     /* value = 0. No parameters in secondary request. */
 WORD     smb_dscnt;      /* number of data bytes being sent this buffer */
 WORD     smb_dsoff;      /* offset (from start of SMB hdr) to data bytes */
 WORD     smb_dsdisp;     /* byte displacement for these data bytes */
 WORD     smb_fid;        /* value = 0xffff no handle on request */
 WORD     smb_bcc;        /* total bytes (including pad bytes) following */
  BYTE    smb_pad[];      /* (optional) to pad to word or dword boundary */
  BYTE    smb_data[*];    /* data bytes (* = value of smb_dscnt) */
```

First Response Format :

```
  BYTE    smb_wct;          /* value = 10 */
 WORD     smb_tprcnt;       /* value = 6 */
 WORD     smb_tdrcnt;       /* value = total length of return data buffer  */
 WORD     smb_rsvd;         /* reserved */
 WORD     smb_prcnt;        /* parameter bytes returned in this buffer */
 WORD     smb_proff;        /* offset (from start of SMB hdr) to param bytes */
 WORD     smb_prdisp;       /* value = 0 byte displacement for param bytes */
 WORD     smb_drcnt;        /* data bytes returned in this buffer */
 WORD     smb_droff;        /* offset (from start of SMB hdr) to data bytes */
 WORD     smb_drdisp;       /* byte displacement for these data bytes */
  BYTE    smb_suwcnt;       /* value = 0 no set up return words */
  BYTE    smb_rsvd1;        /* reserved (pad above to word) */
 WORD     smb_bcc;          /* total bytes (including pad bytes) following */
  BYTE    smb_pad[];        /* (optional) to pad to word or dword boundary */
  BYTE    smb_param[*];     /* The parmater block for the
                             * TRANSACT2_FINDNOTIFYFIRST function response is
                             * the find first specific return
                             * information in the following format. */
           WORD             findnfirst_handle;       /* Mointor handle */
           WORD             findnfirst_changecount;  /* Number of changes which
                                                      * occured within timeout */
           WORD             findnfirst_offerror;     /* error offset if EA error */
  BYTE    smb_pad1[];       /* (optional) to pad to word or dword boundary */
  BYTE    smb_data[*];      /* return data bytes (* = value of smb_dscnt) */
                            /* The data block contains the level dependent
                             * information about the changes which occurred
                             */
```

Subsequent Response Format :

```
 BYTE   smb_wct;       /* value = 10 */
 WORD   smb_tprcnt;    /* value = 6 */
 WORD   smb_tdrcnt;    /* value = total length of return data buffer  */
 WORD   smb_rsvd;      /* reserved */
 WORD   smb_prcnt;     /* value = 0 */
 WORD   smb_proff;     /* value = 0 */
 WORD   smb_prdisp;    /* value = 0 */
 WORD   smb_drcnt;     /* data bytes returned in this buffer */
 WORD   smb_droff;     /* offset (from start of SMB hdr) to data bytes */
 WORD   smb_drdisp;    /* byte displacement for these data bytes */
 BYTE   smb_suwcnt;    /* value = 0 no set up return words */
 BYTE   smb_rsvd1;     /* reserved (pad above to word) */
 WORD   smb_bcc;       /* total bytes (including pad bytes) following */
 BYTE   smb_pad1[];    /* (optional) to pad to word or dword boundary */
 BYTE   smb_data[*];   /* return data bytes (* = value of smb_dscnt) */
                       /* The data block contains the level dependent
                        * information about the changes which occurred
                        */
```

6.0.1.1.13. TRANSACT2_FINDNOTIFYNEXT

The function code TRANSACT2_FINDNOTIFYNEXT in smb_setup[0] in the primary TRANSACT2 request identifies a request to continue monitoring changes to a file or directory specified by a TRANSACT_FINDNOTIFYFIRST request.

Primary Request Format:

```
  BYTE    smb_wct;          /* value = 15 */
  WORD    smb_tpscnt;       /* value = 4, total param bytes being sent */
  WORD    smb_tdscnt;       /* total number of data bytes being sent */
  WORD    smb_mprcnt;       /* value = maximum return parameter length */
  WORD    smb_mdrcnt;       /* value = maximum return data length */
  BYTE    smb_msrcnt;       /* value = 0. No setup words to return */
  BYTE    smb_rsvd;         /* reserved (pad above to word) */
  WORD    smb_flags;        /* additional information:
                                bit 0 - 0
                                bit 1 - 0 */
 DWORD    smb_timeout;      /* Duration of monitor period */
  WORD    smb_rsvd1;        /* reserved */
  WORD    smb_pscnt;        /* value = tpscnt, parms must be in primary request */
  WORD    smb_psoff;        /* offset (from start of SMB hdr to parameter bytes */
  WORD    smb_dscnt;        /* number of data bytes being sent this buffer */
  WORD    smb_dsoff;        /* offset (from start of SMB hdr) to data bytes */
  BYTE    smb_suwcnt;       /* value = 1 */
  BYTE    smb_rsvd2;        /* reserved (pad above to word) */
  WORD    smb_setup1;       /* value = 12  :- TRANSACT2_FINDNOTIFYNEXT */
  WORD    smb_bcc;          /* total bytes (including pad bytes) following */
  BYTE    smb_pad[];        /* (optional) to pad to word or dword boundary */
  BYTE    smb_param[*];     /* The parmater block for the
                             * TRANSACT2_FINDNOTIFYNEXT function
                             * is the find next specific information
                             * in the following format. */
           WORD             findnnext_DirHandle;      /* Directory monitor handle */
           WORD             findnnext_ChangeCount;    /* Number of changes to wait for */
  BYTE    smb_pad1[];       /* (optional) to pad to word or dword boundary */
  BYTE    smb_data[*];      /* Additional level dependent monitor
                             * information.
                             */
```

Secondary Request Format (more data - may be zero or more of these):

```
 BYTE    smb_wct;        /* value = 9 */
 WORD    smb_tpscnt;     /* value = 4 total parmeter bytes sent */
 WORD    smb_tdscnt;     /* total number of data bytes being sent */
 WORD    smb_pscnt;      /* value = 0. All params in primary request */
 WORD    smb_psoff;      /* value = 0. No parameters in secondary request. */
 WORD    smb_psdisp;     /* value = 0. No parameters in secondary request. */
 WORD    smb_dscnt;      /* number of data bytes being sent this buffer */
 WORD    smb_dsoff;      /* offset (from start of SMB hdr) to data bytes */
 WORD    smb_dsdisp;     /* byte displacement for these data bytes */
 WORD    smb_fid;        /* search handle */
 WORD    smb_bcc;        /* total bytes (including pad bytes) following */
 BYTE    smb_pad[];      /* (optional) to pad to word or dword boundary */
 BYTE    smb_data[*];    /* data bytes (* = value of smb_dscnt) */
```

First Response Format :

```
 BYTE    smb_wct;          /* value = 10 */
 WORD    smb_tprcnt;       /* value = 4 */
 WORD    smb_tdrcnt;       /* value = total length of return data buffer  */
 WORD    smb_rsvd;         /* reserved */
 WORD    smb_prcnt;        /* parameter bytes returned in this buffer */
 WORD    smb_proff;        /* offset (from start of SMB hdr) to param bytes */
 WORD    smb_prdisp;       /* value = 0 byte displacement for param bytes */
 WORD    smb_drcnt;        /* data bytes returned in this buffer */
 WORD    smb_droff;        /* offset (from start of SMB hdr) to data bytes */
 WORD    smb_drdisp;       /* byte displacement for these data bytes */
 BYTE    smb_suwcnt;       /* value = 0 no set up return words */
 BYTE    smb_rsvd1;        /* reserved (pad above to word) */
 WORD    smb_bcc;          /* total bytes (including pad bytes) following */
 BYTE    smb_pad[];        /* (optional) to pad to word or dword boundary */
 BYTE    smb_param[*];     /* The parmater block for the
                            * TRANSACT2_FINDNOTIFYNEXT function
                            * response is the find notify next specific return
                            * information in the following format. */
         WORD              findnnext_changecount;  /* Number of changes which
                                                    * during the monitor period. */
         WORD              findnnext_offerror;
 BYTE    smb_pad1[];       /* (optional) to pad to word or dword boundary */
 BYTE    smb_data[*];      /* return data bytes (* = value of smb_dscnt) */
                           /* The data block contains the level dependent
                            * information about the changes which occurred.
                            */
```

Subsequent Response Format :

```
 BYTE    smb_wct;        /* value = 10 */
WORD     smb_tprcnt;     /* value = 4 */
WORD     smb_tdrcnt;     /* value = total length of return data buffer  */
WORD     smb_rsvd;       /* reserved */
WORD     smb_prcnt;      /* value = 0 */
WORD     smb_proff;      /* value = 0 */
WORD     smb_prdisp;     /* value = 0 */
WORD     smb_drcnt;      /* data bytes returned in this buffer */
WORD     smb_droff;      /* offset (from start of SMB hdr) to data bytes */
WORD     smb_drdisp;     /* byte displacement for these data bytes */
 BYTE    smb_suwcnt;     /* value = 0 no set up return words */
 BYTE    smb_rsvd1;      /* reserved (pad above to word) */
WORD     smb_bcc;        /* total bytes (including pad bytes) following */
 BYTE    smb_pad1[];     /* (optional) to pad to word or dword boundary */
 BYTE    smb_data[*];    /* return data bytes (* = value of smb_dscnt) */
                         /* The data block contains the level dependent
                          * information about the changes which occurred.
                          */
```

6.0.1.1.14. TRANSACT2_MKDIR

The function code TRANSACT2_MKDIR in smb_setup[0] in the primary TRANSACT2 requests identifies a request to create a directory with extended attributes.

Primary Request Format:

```
  BYTE    smb_wct;          /* value = 15 */
  WORD    smb_tpscnt;       /* value = total number of param bytes being sent */
  WORD    smb_tdscnt;       /* total number of data bytes being sent */
  WORD    smb_mprcnt;       /* value = maximum return parameter length */
  WORD    smb_mdrcnt;       /* value = 0. No data returned */
  BYTE    smb_msrcnt;       /* value = 0. No setup words to return */
  BYTE    smb_rsvd;         /* reserved (pad above to word) */
  WORD    smb_flags;        /* additional information:
                             *    bit 0 - 0
                             *    bit 1 - 0 */
 DWORD    smb_timeout;      /* value = 0. Not used for mkdir */
  WORD    smb_rsvd1;        /* reserved */
  WORD    smb_pscnt;        /* value = tpscnt, parms must be in primary request */
  WORD    smb_psoff;        /* offset (from start of SMB Hdr to parameter bytes */
  WORD    smb_dscnt;        /* number of data bytes being sent this buffer */
  WORD    smb_dsoff;        /* offset (from start of SMB hdr) to data bytes */
  BYTE    smb_suwcnt;       /* value = 1 */
  BYTE    smb_rsvd2;        /* reserved (pad above to word) */
  WORD    smb_setup1;       /* value = 13  :- TRANSACT2_MKDIR */
  WORD    smb_bcc;          /* total bytes (including pad bytes) following */
  BYTE    smb_pad[];        /* (optional) to pad to word or dword boundary */
  BYTE    smb_param[*];     /* The parmater block for the
                             * TRANSACT2_MKDIR function is
                             * the mkdir specific information
                             * in the following format. */
          DWORD              mkdir_rsvd;       /* Reserved. Must be zero. */
          BYTE               mkdir_dirname[];  /* Directory name */
  BYTE    smb_pad1[];       /* (optional) to pad to word or dword boundary */
  BYTE    smb_data[*];      /* FEAList structure for the directory
                             * to be created */
```

Secondary Request Format (more data - may be zero or more of these):

```
  BYTE    smb_wct;        /* value = 9 */
  WORD    smb_tpscnt;     /* totalnumber of parameter bytes being sent */
  WORD    smb_tdscnt;     /* total number of data bytes being sent */
  WORD    smb_pscnt;      /* value = 0. All params in primary request */
  WORD    smb_psoff;      /* value = 0. No parameters in secondary request. */
  WORD    smb_psdisp;     /* value = 0. No parameters in secondary request. */
  WORD    smb_dscnt;      /* number of data bytes being sent this buffer */
  WORD    smb_dsoff;      /* offset (from start of SMB hdr) to data bytes */
  WORD    smb_dsdisp;     /* byte displacement for these data bytes */
  WORD    smb_fid;        /* value = 0xffff, no handle on request */
  WORD    smb_bcc;        /* total bytes (including pad bytes) following */
  BYTE    smb_pad[];      /* (optional) to pad to word or dword boundary */
  BYTE    smb_data[*];    /* data bytes (* = value of smb_dscnt) */
```

Response Format (one only):

```
 BYTE    smb_wct;          /* value = 10 */
 WORD    smb_tprcnt;       /* value = 2 */
 WORD    smb_tdrcnt;       /* value = 0 no data bytes  */
 WORD    smb_rsvd;         /* reserved */
 WORD    smb_prcnt;        /* value = 2, parameter bytes being returned */
 WORD    smb_proff;        /* offset (from start of SMB hdr) to param bytes */
 WORD    smb_prdisp;       /* value = 0 byte displacement for param bytes */
 WORD    smb_drcnt;        /* value = 0 no data bytes */
 WORD    smb_droff;        /* value = 0 no data bytes */
 WORD    smb_drdisp;       /* value = 0 no data bytes */
 BYTE    smb_suwcnt;       /* value = 0 no set up return words */
 BYTE    smb_rsvd1;        /* reserved (pad above to word) */
 WORD    smb_bcc;          /* total bytes (including pad bytes) following */
 BYTE    smb_pad[];        /* (optional) to pad to word or dword boundary */
 BYTE    smb_param[*];     /* The parmater block for the
                            * TRANSACT2_MKDIR function response
                            * is the mkdir specific return
                            * information in the following format. */
         WORD               mkdir_offerror;    /* offset into FEAList data of first
                                                * error which occured while setting
                                                * the extended attributes. */
```

6.0.2. FIND NOTIFY CLOSE

Request Format:

```
  BYTE    smb_wct;       /* value = 1 */
 WORD     smb_handle;    /* Find notify handle */
 WORD     smb_bcc;       /* value = 0 */
```

Response Format:

```
  BYTE    smb_wct;    /* value = 0 */
 WORD     smb_bcc;    /* value = 0 */
```

Service:

The Find Notify Close protocol closes the association between a directory handle returned following a resourse monitor established using a TRANSACT2_FINDNOTIFYFIRST request to the server and the resulting system directory monitor. This request allows the server to free any resources held in support of the open handle.

The Find Close protocol is used to match the DosFindNotifyClose OS/2 system call.

Find Notify Close may generate the following errors.

Error Class ERRDOS

ERRbadfid
<implementation specific>

Error Class ERRSRV

ERRerror
ERRinvnid
<implementation specific>

Error Class ERRHRD

<implementation specific>

6.0.3. FIND CLOSE

Request Format:

```
  BYTE    smb_wct;                /* value = 1 */
  WORD    smb_handle; /* Find handle */
  WORD    smb_bcc;                /* value = 0 */
```

Response Format:

```
  BYTE    smb_wct;    /* value = 0 */
  WORD    smb_bcc;    /* value = 0 */
```

Service:

The Find Close protocol closes the association between a search handle returned following a successful FIND FIRST request sent to the server using the TRANSACT2 protocol and the resulting system file search. This request allows the server to free any resources held in support of the open handle.

The Find Close protocol is used to match the DosFindFirst2 OS/2 system call.

Find Close may generate the following errors.

Error Class ERRDOS

ERRbadfid
<implementation specific>

Error Class ERRSRV

ERRerror
ERRinvnid
<implementation specific>

Error Class ERRHRD

<implementation specific>

6.0.4. USER LOGOFF and X

Request Format:

```
  BYTE   smb_wct;    /* value = 2 */
  BYTE   smb_com2;   /* secondary (X) command, 0xFF = none */
  BYTE   smb_reh2;   /* reserved (must be zero) */
  WORD   smb_off2;   /* offset (from SMB hdr start) to next cmd (@smb_wct) */
  WORD   smb_bcc;    /* value = 0 */
```

Response Format:

```
  BYTE   smb_wct;    /* value = 2 */
  BYTE   smb_com2;   /* secondary (X) command, 0xFF = none */
  BYTE   smb_res2;   /* reserved (pad to word) */
  WORD   smb_off2;   /* offset (from SMB hdr start) to next cmd (@smb_wct) */
  WORD   smb_bcc;    /* value = 0 */
```

Service definition:

This protocol is used to "Log Off" the user (identified by the UID value in smb_uid) previously "Logged On" via the Session Set Up protocol.

The server will remove this UID from its list of valid UIDs for this session. Any subsequent protocol containing this UID (in smb_uid) received (on this session) will be returned with an access error.

Another Session Set Up ("User Logon") must be sent in order to reenstate the user on the session.

Session Termination also causes the UIDs registered on the session to be invalidated. When the session is reestablished, Session Setup request(s) must again be used to validate each user.

The following are the only valid protocol request commands for smb_com2 (X) for User Logoff and X:

SESSION SET UP and X

User Logoff may generate the following errors.

Error Class ERRDOS

<implementation specific>

Error Class ERRSRV

<implementation specific>

Error Class ERRHRD

<implementation specific>

7. DATA DEFINITIONS

7.1. COMMAND CODES

The command codes are unchanged for commands that are common with the Core File Sharing Protocol.

The following values have been assigned for the "core" protocol commands.

```
#define SMBmkdir      0x00   /* create directory */
#define SMBrmdir      0x01   /* delete directory */
#define SMBopen       0x02   /* open file */
#define SMBcreate     0x03   /* create file */
#define SMBclose      0x04   /* close file */
#define SMBflush      0x05   /* flush file */
#define SMBunlink     0x06   /* delete file */
#define SMBmv         0x07   /* rename file */
#define SMBgetatr     0x08   /* get file attributes */
#define SMBsetatr     0x09   /* set file attributes */
#define SMBread       0x0A   /* read from file */
#define SMBwrite      0x0B   /* write to file */
#define SMBlock       0x0C   /* lock byte range */
#define SMBunlock     0x0D   /* unlock byte range */
#define SMBctemp      0x0E   /* create temporary file */
#define SMBmknew      0x0F   /* make new file */
#define SMBchkpth     0x10   /* check directory path */
#define SMBexit       0x11   /* process exit */
#define SMBlseek      0x12   /* seek */
#define SMBtcon       0x70   /* tree connect */
#define SMBtdis       0x71   /* tree disconnect */
#define SMBnegprot    0x72   /* negotiate protocol */
#define SMBdskattr    0x80   /* get disk attributes */
#define SMBsearch     0x81   /* search directory */
#define SMBsplopen    0xC0   /* open print spool file */
#define SMBsplwr      0xC1   /* write to print spool file */
#define SMBsplclose   0xC2   /* close print spool file */
#define SMBsplretq    0xC3   /* return print queue */
#define SMBsends      0xD0   /* send single block message */
#define SMBsendb      0xD1   /* send broadcast message */
#define SMBfwdname    0xD2   /* forward user name */
#define SMBcancelf    0xD3   /* cancel forward */
#define SMBgetmac     0xD4   /* get machine name */
#define SMBsendstrt   0xD5   /* send start of multi-block message */
#define SMBsendend    0xD6   /* send end of multi-block message */
#define SMBsendtxt    0xD7   /* send text of multi-block message */
```

The commands added by the LANMAN 1.0 Extended File Sharing Protocol have the following command codes:

```
#define SMBlockread      0x13   /* lock then read data */
#define SMBwriteunlock   0x14   /* write then unlock data */
#define SMBreadBraw      0x1A   /* read block raw */
#define SMBreadBmpx      0x1B   /* read block multiplexed */
#define SMBreadBs        0x1C   /* read block (secondary response) */
#define SMBwriteBraw     0x1D   /* write block raw */
#define SMBwriteBmpx     0x1E   /* write block multiplexed */
#define SMBwriteBs       0x1F   /* write block (secondary request) */
#define SMBwriteC        0x20   /* write complete response */
#define SMBsetattrE      0x22   /* set file attributes expanded */
#define SMBgetattrE      0x23   /* get file attributes expanded */
#define SMBlockingX      0x24   /* lock/unlock byte ranges and X */
#define SMBtrans         0x25   /* transaction - name, bytes in/out */
#define SMBtranss        0x26   /* transaction (secondary request/response) */
#define SMBioctl         0x27   /* IOCTL */
#define SMBioctls        0x28   /* IOCTL  (secondary request/response) */
#define SMBcopy          0x29   /* copy */
#define SMBmove          0x2A   /* move */
#define SMBecho          0x2B   /* echo */
#define SMBwriteclose    0x2C   /* Write and Close */
#define SMBopenX         0x2D   /* open and X */
#define SMBreadX         0x2E   /* read and X */
#define SMBwriteX        0x2F   /* write and X */
#define SMBsesssetup     0x73   /* Session Set Up & X (including User Logon) */
#define SMBtconX         0x75   /* tree connect and X */
#define SMBffirst        0x82   /* find first */
#define SMBfunique       0x83   /* find unique */
#define SMBfclose        0x84   /* find close */
#define SMBinvalid       0xFE   /* invalid command */
```

The commands added by the LANMAN 2.0 Extended File Sharing Protocol have the following command codes:

```
#define SMBtrans2       0x32   /* transaction2 - function, byte in/out */
#define SMBtranss2      0x33   /* transaction2 (secondary request/response*/
#define SMBfindclose    0x34   /* find close */
#define SMBfindnclose   0x35   /* find notify close */
#define SMBuloggoffX    0x74   /* User logoff and X */
```

7.2. ERROR CLASSES AND CODES

The error class and code lists in the section include all classes and codes generated by the Core File Sharing Protocol. Errors listed here are intended to provide a finer granularity of error conditions. These lists are not complete.

The following error classes may be returned by the protocol elements defined in this document.

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.
ERRXOS	0x04	Reserved for XENIX.
ERRRMX1	0xE1	Reserved for iRMX
ERRRMX2	0xE2	Reserved for iRMX
ERRRMX3	0xE3	Reserved for iRMX
ERRCMD	0xFF	Command was not in the "SMB" format.

The following error codes may be generated with the SUCCESS error class.

SUCCESS	0	The request was successful.

The following error codes may be generated with the ERRDOS error class. The XENIX errors equivalent to each of these errors are noted at the end of the error description. NOTE - When the extended protocol (LANMAN 1.0) has been negotiated, all of the error codes below may be generated plus any of the new error codes defined for OS/2 (see OS/2 operating system documentation for complete list of OS/2 error codes). When only "core" protocol has been negotiated, the server must map additional OS/2 (or OS/2 like) errors to the errors listed below.

The following error codes may be generated with the ERRDOS error class.

ERRbadfunc	1	Invalid function. The server OS 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. [EINVAL]
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. [ENOENT]
ERRnofids	4	Too many open files. The server has no file handles (FIDs) available. [EMFILE]
ERRnoaccess	5	Access denied, the requester's context does not permit the requested function. This includes the following conditions. [EPERM] 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. [EBADF]
ERRbadmcb	7	Memory control blocks destroyed. [EREMOTEIO]
ERRnomem	8	Insufficient server memory to perform the requested function. [ENOMEM]
ERRbadmem	9	Invalid memory block address. [EFAULT]
ERRbadenv	10	Invalid environment. [EREMOTEIO]
ERRbadformat	11	Invalid format. [EREMOTEIO]
ERRbadaccess	12	Invalid open mode.
ERRbaddata	13	Invalid data (generated only by IOCTL calls within the server). [E2BIG]
ERR	14	reserved
ERRbaddrive	15	Invalid drive specified. [ENXIO]
ERRremcd	16	A Delete Directory request attempted to remove the server's current directory. [EREMOTEIO]
ERRdiffdevice	17	Not same device (e.g., a cross volume rename was attempted) [EXDEV]
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. [ETXTBSY]
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. [EDEADLOCK]
ERRfilexists	80	The file named in a Create Directory, Make New File or Link request already exists. The error may also be generated in the Create and Rename transaction. [EEXIST]
ERRbadpipe	230	Pipe invalid.
ERRpipebusy	231	All instances of the requested pipe are busy.
ERRpipeclosing	232	Pipe close in progress.
ERRnotconnected	233	No process on other end of pipe.
ERRmoredata	234	There is more data to be returned.

The following error codes may be generated with the ERRSRV error class.

ERRerror	1	Non-specific error code. It is returned under the following conditions: resource other than disk space exhausted (e.g. TIDs) first command on VC was not negotiate multiple negotiates attempted internal server error [ENFILE]
ERRbadpw	2	Bad password - name/password pair in a Tree Connect or Session Setup are invalid.
ERRbadtype	3	reserved
ERRaccess	4	The requester does not have the necessary access rights within the specified context for the requested function. The context is defined by the TID or the UID. [EACCES]
ERRinvnid	5	The tree ID (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 file handle (FID) and pathname parameters contained an invalid combination of values.
ERRreserved	68	reserved.
ERRbadpermits	69	The access permissions specified for a file or directory are not a valid combination. The server cannot set the requested attribute.
ERRreserved	70	reserved.
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 ID on this session.
ERRusempx	250	Temp unable to support Raw, use MPX mode.
ERRusestd	251	Temp unable to support Raw, use standard read/write.
ERRcontmpx	252	(reserved) continue in MPX mode.
ERRreserved	253	reserved.
ERRreserved	254	reserved.
ERRnosupport	0xFFFF	Function not supported.

The following error codes may be generated with the ERRHRD error class. The XENIX errors equivalent to each of these errors are noted at the end of the error description.

ERRnowrite	19	Attempt to write on write-protected diskette. [EROFS]
ERRbadunit	20	Unknown unit. [ENODEV]
ERRnotready	21	Drive not ready. [EUCLEAN]
ERRbadcmd	22	Unknown command.
ERRdata	23	Data error (CRC). [EIO]
ERRbadreq	24	Bad request structure length. [ERANGE]
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. [ETXTBSY]
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. [EDEADLOCK]
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.