--

--

Microsoft Networks/OpenNET

FILE SHARING PROTOCOL

INTEL Part Number 138446

Document Version 2.0

November 7, 1988

Microsoft Corporation
Intel Corporation

INTEL PN 138446

1. Introduction

This document describes the MSNET/PCNET file sharing protocol. Systems can use these protocols to obtain or provide remote file services in a network environment. These protocols are designed to allow systems to transparently access files which reside on remote systems. Items which are mapped into the file space (such as UNIX style "device special files") are also transparently shared by these protocols.

When two machines first come into network contact they may negotiate the use of a higher level "Extension Protocol". For example, two MS-DOS machines would agree to use the MS-DOS-specific protocol extensions. These extensions can include both new messages as well as changes to the fields and semantics of existing messages. The "Core/Extension Protocol" definition allows a system to communicate at a strong, functional level with other "core" machines, and to communicate in full transparent detail to its "brother" systems. The ability to negotiate the protocol used across a given connection is also used, in those cases where multiple versions of a protocol exist, to ensure that only compatible versions of the protocol are used.

This document assumes the existence of, but does not describe, a lower level set of protocols that provide for virtual circuits and transport between clients and servers. Further, it does not discuss the mechanism used to "identify" and "locate" a correspondent in order to establish said virtual circuit. The details of virtual circuit support for MS-DOS are described in the document "Transport Layer Interface".

2. Message Format

Every message has a common format. The following C-language style definition shows that format.

```
BYTE    smb_idf[4];    /* contains 0xFF, 'SMB' */
BYTE    smb_com;       /* command code */
BYTE    smb_rcls;      /* error code class */
BYTE    smb_reh;       /* reserved (contains AH if DOS INT-24 ERR) */
WORD    smb_err;       /* error code */
BYTE    smb_reb;       /* reserved */
WORD    smb_res[7];    /* reserved */
WORD    smb_tid;       /* tree id #  */
WORD    smb_pid;       /* caller's process id # */
WORD    smb_uid;       /* user id # */
WORD    smb_mid;       /* mutiplex id #  */
BYTE    smb_wct;       /* count of parameter words */
WORD    smb_vwv[];     /* variable # words of params */
WORD    smb_bcc;       /* # bytes of data following */
BYTE    smb_data[];    /* data bytes */
```

A BYTE is an octet.
A WORD is two bytes.
The bytes within a word are ordered such that the low byte precedes the high byte.

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 sub-tree. (see below)

smb_pid:caller's process id. Generated by the consumer to uniquely identify a process within the consumers system.

smb_mid:this field is reserved for multiplexing multiple messages on a single Virtual Circuit (VC). A response message will always contain the same value as the corresponding request message. This initial version of the core protocol will not support multiplexing within a VC. Only one request at a time may be outstanding on any VC.

3. Architectural Model

The Network File Access system described in this document deals with two types of systems on the network -- consumers and servers. A consumer is a system that requests network file services and a server is a system that delivers network file services. Consumers and servers are logical systems; a consumer and server may coexist in a single physical system.

Consumers are responsible for directing their requests to the appropriate server. The network addressing mechanism or naming convention through which the server is identified is outside the scope of this document.

Each server makes available to the network a self-contained file structure. There are no storage or service dependencies on any other servers. A file must be entirely contained by a single server.

The core file sharing protocol requires server authentication of users before file accesses are allowed. Each server processor authenticates its own users. A user must "login" to each server that it wishes to access.

This authentication model assumes that the LAN connects autonomous systems that are willing to make some subset of their local files available to remote users.

The following environments exist in the core file sharing protocol environment.

a) Virtual Circuit Environment. This consists of one VC established between a consumer system and server system. Consumers may have only a single request active on any VC at any time, i.e., a second request cannot be initiated until the response to the first has been received. A VC is formed using transport services.

b) Logon Environment. This is represented by a Tree ID (TID). A TID uniquely identifies a file sharing connection between a consumer and server. It also identifies the scope and type of accesses allowed across the connection. With the exception of the Tree Connect and Negotiate commands, the TID field in a message must always contain a valid TID. There may be any number of file sharing connections per VC.

c) Process Environment. This is represented by a process ID (PID). A PID uniquely identifies a consumer process within a given VC environment.

d) File Environment. This is represented by a File Handle (FID). A FID identifies an open file and is unique within a given VC environment.

When one of these environments is terminated, all environments contained within it will be terminated. For example, if a VC is terminated all PIDs, TIDs and FIDs within it will be invalidated.

3.1. Process Management

How and when servers create and destroy processes is, of course, an implementation issue and there is no requirement that this be tied in any way to the consumer's process management. However, it is necessary for the server to be aware of the consumer's process management activities as files are accessed on behalf of consumer processes. Therefore the file sharing protocol includes appropriate notifications.

All messages, except Negotiate, include a process ID (PID) to indicate which user process initiated a request. Consumers inform servers of the creation of a new process by simply introducing a new PID into the dialogue. Process destruction must be explicitly indicated and the "Process Exit" command is provided for this purpose. The consumer must send a Process Exit command whenever a user process is destroyed. This enables the server to free any resources (e.g., locks) reserved by that process as well as perform any local process management activities that its implementation might require.

4. File Sharing Connections

The networks using this file sharing protocol will contain not only multi-user systems with user based protection models, but single-user systems that have no concept of user-ids or permissions. Once these machines are connected to the network, however, they are in a multi-user environment and need a method of access control. First, unprotected machines need to be able to provide some sort of bona-fides to other net machines which do have permissions, secondly unprotected machines need to control access to their files by others.

This protocol defines a mechanism that enables the network software to provide the protection where it is missing from the operating system, and supports user based protection where it is provided by the operating system. The mechanism also allows machines with no concept of user-id to demonstrate access authorization to machines which do have a permission mechanism. Finally, the permission protocol is designed so that it can be omitted if both machines share a common permission mechanism.

This protocol, called the "tree connection" protocol, does not specify a user interface. A possible user interface will be described by way of illustration.

4.1. Unprotected Server Machines

The following examples apply to access to serving systems which do not have a permission mechanism.

a) NET SHARE

By default (on unprotected machines) all network requests are refused as unauthorized. Should a user wish to allow access to some or all of his files he offers access to an arbitrary set of subtrees by specifying each subtree and a password.

Examples:

```
NET SHARE  \dir1  "bonzo"
```

assign password "bonzo" to all files within directory "dir1" and its subdirectories.

```
NET SHARE  \        " "      RO

NET SHARE  \work  "flipper"  RW
```

offer read-only access to everything (all files are within the root directory or its subdirectories) Offer read-write access to all files within the \work directory and its subdirectories.

b) NET USE

Other users can gain access to one or more offered subtrees via the NET USE command. Once the NET USE command is issued the user can access the files freely without further special requirements.

Examples:

```
1. NET USE  \\machine-name\dir1  "bonzo"
```

now any pathname starting with \\machine-name\dir1 is valid.

```
2. NET USE  \\machine-name\

3. NET USE  \\machine-name\work  "flipper"
```

Now any read request to any file on that machine is valid. Read-write requests only succeed to files whose pathnames start with \\machine-name\work

The requester must remember the machine-name pathname prefix combination supplied with the NET USE request and associate it with the index value returned by the server. Subsequent requests using this index must include only the pathname relative to the connected subtree as the server treats the subtree as the root directory.

When the requester has a file access request for the server, it looks through its list of prefixes for that machine and selects the most specific (the longest) match. It then includes the index associated with this prefix in his request along with the remainder of the pathname.

Note that one always offers a directory and all files underneath that directory are then affected. If a particular file is within the range of multiple offer ranges, connecting to any of the offer ranges gains access to the file with the permissions specified for the offer named in the NET USE. The server will not check for nested directories with more restrictive permissions.

4.2. Protected Server Machines

Servers with user based file protection schemes will interpret the Tree Connect command slightly differently from systems with file oriented file protection schemes. They interpret the "name" parameter as a username rather than a pathname. When this request is received, the username is validated and a TID representing that authenticated instance of the user is returned. This TID must be included in all further requests made on behalf of the user.

The permission-based system need not execute a NET SHARE command; instead it sets up name/password (or whatever) information in its user definition files. The accessing user would type

    NET USE \\machine-name\account-name <password>

and thereby "login" to the serving machine. He need not specify subtrees and so forth because the account-name/password pair establishes access permissions to everything on that machine.

This variation of Tree Connect is an aspect of the the server's file system. Servers with user based protection schemes will always interpret the name supplied with Tree Connect as a user name. Users of Tree Connect simply provide a "name" and its associated "password"; they do not need to be aware of the server's interpretation of that name. If the name and password are successfully authenticated the caller receives access to the set of files protected by the name in the modes allowed by the server (also determined by the name/password pair).

4.3. Connection Protocols

The NET SHARE command generates no network messages. The server package remembers the pathname prefix and the password.

The NET USE command generates a message containing the path/username and the password. The serving machine verifies the combination and returns an error code or an identifier. The full name (path or user) is included in the Tree Connect request message and the identifier identifying the connection is returned in the smb_tid field. The meaning of this identifier (tid) is server specific; the requester must not associate any specific meaning to it.

The server makes whatever use of the tid field it desires. Normally it is an index into a server table which allows the server to optimize its response.

4.4. Tree Connect

>From Consumer		To Consumer	
smb_com	SMBtcon	smb_com	SMBtcon
smb_wct	0	smb_wct	2
smb_bcc	min=4	smb_vwv[0]	max xmit size
smb_buf[]	ASCII -- 04	smb_vwv[1]	TID
	path/username	smb_bcc	0
	ASCII -- 04		
	password		
	ASCII -- 04		
	dev name		

The device name is either <device>: for block device or LPT1: for a character device.

The path/username must be specified from the network root (including \\). The TID field in the request message is ignored by the server. The maximum transmit size field in the response message indicates the maximum size message that the server can handle. The consumer should not generate messages, nor expect to receive responses, larger than this. This should be constant for a given server.

Tree Connects must be issued for all subtrees accessed, even if they contain a null password.

Tree Connect may generate the following errors:

Error Class ERRDOS:

<implementation specific>

Error Class ERRSRV:

ERRerror
ERRbadpw
ERRinvnetname
<implementation specific>

Error Class ERRHRD:

<implementation specific>

4.5. Tree Disconnect

>From Consumer		To Consumer	
smb_com	SMBtdis	smb_com	SMBtdis
smb_wct	0	smb_wct	0
smb_bcc	0	smb_bcc	0

The file sharing connection identified by the TID is logically disconnected from the server. The TID will be invalidated; it will not be recognized if used by the consumer for subsequent requests.

Tree Disconnect may generate the following errors:

Error Class ERRDOS:

<implementation specific>

Error Class ERRSRV:

ERRinvnid
<implementation specific>

Error Class ERRHRD:

<implementation specific>

5. File Sharing Commands

The message definitions in this section indicate the command code and include the balance of the definition commencing at the field smb_wct. The omitted fields (smb_cls through smb_mid) are constant in the format and meaning defined in Section 1.0. When an error is encountered a server may return only the header portion of the response (i.e., smb_wct and smb_bcc both contain zero). The data objects used by these commands are described in section 6.0.

The use of commands other than those defined in this section will have undefined results.

5.1. Open File

>From Consumer		To Consumer	
smb_com	SMBopen	smb_com	SMBopen
smb_wct	2	smb_wct	7
smb_vwv[0]	r/w/share	smb_vwv[0]	file handle
smb_vwv[1]	attribute	smb_vwv[1]	attribute
smb_bcc	min = 2	smb_vwv[2]	time1 low
smb_buf[]	ASCII -- 04	smb_vwv[3]	time1 high
	file pathname	smb_vwv[4]	file size low
		smb_vwv[5]	file size high
		smb_vwv[6]	access allowed
		smb_bcc	0

This message is sent to obtain a file handle for a data file. The relevant tree id and any necessary additional pathname are passed. The handle returned can be used in subsequent read, write, lock, unlock and close messages. The file size and last modification time are also returned. The r/w/share word controls the mode. The file will be opened only if the requester has the appropriate permissions. The r/w/share word has the following format and values.

r/w/share format: - - - - - - - - rxxx yyyy

where: r = reserved

xxx = 0 -- MS-DOS Compatibility mode (exclusive to a VC, but that VC may have multiple opens). Support of this mode is optional. However, if it is not supported or is mapped to exclusive open modes, some existing MS-DOS applications may not work with network files. If reading map to deny write, otherwise map to deny read/write.

1 -- Deny read/write (exclusive to this open operation).

2 -- Deny write -- other users may access file in READ mode.

3 -- Deny read -- other users may access file in WRITE mode. Support of this mode is optional.

4 -- Deny none -- allow other users to access file in any mode for which they have permission.

yyyy = 0 -- Open file for reading.

1 -- Open file for writing.
2 -- Open file for reading and writing.

rxxx yyyy = 11111111 (hex FF)
FCB open: This type of open will cause an MS-DOS compatibility mode open with the read/write modes set to the maximum permissible, i.e., if the requester can have read and write access on the file, it will be opened in read/write mode.

The response message indicates the access permissions actually allowed in the "access allowed" field. This field may have the following values:

0 = read-only
1 = write-only
2 = read/write

File Sharing Notes:

1. File Handles (FIDs) are contained within the Virtual Circuit (VC) environment. A PID may reference any FID established by itself or any other PID within its VC. The actual accesses allowed through the FID will depend on the open and deny modes specified when the file was opened (see below).
2. The MS-DOS compatibility mode of file open provides exclusion at the VC level. A file open in compatibility mode may be opened (also in compatibility mode) any number of times for any combination of reading and writing (subject to the user's permissions) by any PID within the owning VC. If the first VC has the file open for writing, then the file may not be opened in any way by any PID within another VC. If the first VC has the file open only for reading, then other VCs may open the file, in compatibility mode, for reading. Once multiple VCs have the file open for reading, no VC is permitted to open the file for writing. No VC or PID may open the file in any mode other than compatibility mode.
3. The other file exclusion modes (Deny read/write, Deny write, Deny read, Deny none) provide exclusion at the file level. A file opened in any "Deny" mode may be opened again only for the accesses allowed by the Deny mode (subject to the user's permissions). This is true regardless of the identity of the second opener -- a PID within another VC, a PID within the same VC, or the PID that already has the file open. For example, if a file is open in "Deny write" mode a second open may only obtain read permission to the file.
4. Although FIDs are available to all PIDs on a VC, PIDs other than the owner may not have the full access rights specified in the open mode by the FID's creator. If the open creating the FID specified a deny mode, then any PID using the FID, other than the creating PID, will have only those access rights determined by "anding" the open mode rights and the deny mode rights, i.e., the deny mode is checked on all file accesses. For example, if a file is opened for Read/Write in Deny write mode, then other VC PIDs may only read from the FID and cannot write; if a file is opened for Read in Deny read mode, then the other VC PIDs can neither read nor write the FID.

If a file cannot be opened for any reason, including a conflict of share modes, a reply message indicating the cause of the failure will be returned.

Open may generate the following errors:

Error Class ERRDOS:

ERRbadfile
ERRnofids
ERRnoaccess
ERRshare

Error Class ERRSRV:

ERRerror
ERRaccess
ERRinvnid
ERRinvdevice
<implementation specific>

Error Class ERRHRD:

<implementation specific>

5.2. Create File

>From Consumer		To Consumer	
smb_com	SMBcreate	smb_com	SMBcreate
smb_wct	3	smb_wct	1
smb_vwv[0]	attribute	smb_vwv[0]	file handle
smb_vwv[1]	time low	smb_bcc	0
smb_vwv[2]	time high		
smb_bcc	min = 2		
smb_buf[]	ASCII -- 04		
	file pathname		

This message is sent to create a new data file or truncate an existing data file to length zero, and open the file. The handle returned can be used in subsequent read, write, lock, unlock and close messages.

Unprotected servers will require requesters to have create permission for the subtree containing the file in order to create a new file, or write permission for the subtree in order to truncate an existing one. The newly created file will be opened in compatibility mode with the access mode determined by the containing subtree permissions.

Protected servers will require requesters to have write permission on the file's parent directory in order to create a new file, or write permission on the file itself in order to truncate it. The access permissions granted on a created file will be read/write permission for the creator. Access permissions for truncated files are not modified. The newly created or truncated file is opened in read/write/compatibility mode.

Support of the create time supplied in the request is optional.

Create may generate the following errors:

Error Class ERRDOS:

ERRbadpath
ERRnofids
ERRnoaccess
ERRbadaccess

Error Class ERRSRV:

ERRerror
ERRaccess
ERRinvnid
ERRinvdevice
<implementation specific>

Error Class ERRHRD:

<implementation specific>

5.3. Close File

>From Consumer		To Consumer	
smb_com	SMBclose	smb_com	SMBclose
smb_wct	3	smb_wct	0
smb_vwv[0]	file handle	smb_bcc	0
smb_vwv[1]	time low		
smb_vwv[2]	time high		
smb_bcc	0		

The close message is sent to invalidate a file handle for the requesting process. All locks held by the requesting process on the file will be "unlocked". The requesting process can no longer use the file handle for further file access requests. The new modification time may be passed to the server. Server support of the modification time is optional; it may be ignored.

Close will cause all the file's buffers to be flushed to disk.

Close may generate the following errors:

Error Class ERRDOS:

ERRbadfid
ERRnoaccess

Error Class ERRSRV:

ERRerror
ERRinvdevice
ERRinvnid
<implementation specific>

Error Class ERRHRD:

<implementation specific>

5.4. Flush File

>From Consumer		To Consumer	
smb_com	SMBflush	smb_com	SMBflush
smb_wct	1	smb_wct	0
smb_vwv[0]	file handle	smb_bcc	0
smb_bcc	0		

The flush message is sent to ensure all data and allocation information for the corresponding file has been written to non-volatile storage. When the file handle has a value -1 (hex FFFF) the server will perform a flush for all file handles associated with the consumer's process. The response is not sent until the writes are complete.

Note that this protocol does not require that only the specific file's data be written (flushed). It specifies that "at least" the file's data be written.

Flush may generate the following errors:

Error Class ERRDOS:

ERRbadfid
ERRnoaccess
<implementation specific>

Error Class ERRSRV:

ERRerror
ERRinvdevice
ERRinvnid
<implementation specific>

Error Class ERRHRD:

<implementation specific>

5.5. Read

>From Consumer		To Consumer	
smb_com	SMBread	smb_com	SMBread
smb_wct	5	smb_wct	5
smb_vwv[0]	file handle	smb_vwv[0]	count
smb_vwv[1]	count of bytes	smb_vwv[1-4]	reserved (MBZ)
smb_vwv[2]	offset low	smb_bcc	length of data + 3
smb_vwv[3]	offset high	smb_buf[]	Data Block -- 01
smb_vwv[4]	count left		length of data
smb_bcc	0		data

The read message is sent to read bytes of a data file. The count of bytes field is used to specify the requested number of bytes. The offset field specifies the offset in the file of the first byte to be read. The count left field is advisory. If the value is not zero, then it is taken as an estimate of the total number of bytes that will be read -- including those read by this request. This additional information may be used by the server to optimize buffer allocation or read-ahead.

The count field in the response message indicates the number of bytes actually being returned. The count returned may be less than the count requested only if a read specifies bytes beyond the current file size. In this case only the bytes that exist are returned. A read completely beyond the end of file will result in a response of length zero. This is the only circumstance when a zero length response is generated. A count returned which is less than the count requested is the end of file indicator.

If a Read requests more data than can be placed in a message of the max-xmit-size for the TID specified, the server will abort the virtual circuit to the consumer.

Read may generate the following errors:

Error Class ERRDOS:

ERRnoaccess
ERRbadfid

Error Class ERRSRV:

ERRerror
ERRinvdevice
ERRinvnid
<implementation specific>

Error Class ERRHRD:

<implementation specific>

5.6. Write

>From Consumer		To Consumer	
smb_com	SMBwrite	smb_com	SMBwrite
smb_wct	5	smb_wct	1
smb_vwv[0]	file handle	smb_vwv[0]	count
smb_vwv[1]	count of bytes	smb_bcc	0
smb_vwv[2]	offset low		
smb_vwv[3]	offset high		
smb_vwv[4]	count left		
smb_bcc	length of data + 3		
smb_buf[]	Data Block -- 01		
	length of data		
	data		

The write message is sent to write bytes into a data file. The count of bytes field specifies the number of bytes to be written. The offset field specifies the offset in the file of the first byte to be written. The count left field is advisory. If the value is not zero, then it is taken as an estimate of the number of bytes that will be written -- including those written by this request. This additional information may be used by the server to optimize buffer allocation.

The count field in the response message indicates the actual number of bytes written, and for successful writes will always equal the count in the request message. If the number of bytes written differs from the number requested and no error is indicated, then the server has no disk space available with which to satisfy the complete write.

When a write specifies a byte range beyond the current end of file, the file will be extended. Any bytes between the previous end of file and the requested offset will be set to zero (ASCII nul).

When a write specifies a length of zero, the file will be truncated to the length specified by the offset.

If a Write sends a message of length greater than the max-xmit-size for the TID specified, the server will abort the virtual circuit to the consumer.

Write may generate the following errors:

Error Class ERRDOS:

ERRnoaccess
ERRbadfid

Error Class ERRSRV:

ERRerror
ERRinvdevice
ERRinvnid
<implementation specific>

Error Class ERRHRD:

<implementation specific>

5.7. Seek

>From Consumer		To Consumer	
smb_com	SMBlseek	smb_com	SMBlseek
smb_wct	4	smb_wct	2
smb_vwv[0]	file handle	smb_vwv[0]	offset-low
smb_vwv[1]	mode	smb_vwv[1]	offset-high
smb_vwv[2]	offset-low	smb_bcc	0
smb_vwv[2]	offset-high		
smb_bcc	min = 0		

The seek message is sent to set the current file pointer for the requesting process. The starting point of the seek is set by the "mode" field in the request. This may have the following values:

0 = seek from start of file
1 = seek from current file pointer
2 = seek from end of file

The response returns the new file pointer expressed as the offset from the start of the file, and may be beyond the current end of file. An attempt to seek to before the start of file set the file pointer to start of file.

Note: the "current file pointer" at the start of this command reflects the offset plus data length specified in the previous read, write or seek request, and the pointer set by this command will be replaced by the offset specified in the next read, write or seek command.

Seek may generate the following errors:

Error Class ERRDOS:

ERRnoaccess
Errbadfid
<implementation specific>

Error Class ERRSRV:

ERRerror
ERRinvnid
<implementation specific>

Error Class ERRHRD:

<implementation specific>

5.8. Create Directory

>From Consumer		To Consumer	
smb_com	SMBmkdir	smb_com	SMBmkdir
smb_wct	0	smb_wct	0
smb_bcc	min = 2	smb_bcc	0
smb_buf[]	ASCII -- 04		
	dir pathname		

The create directory message is sent to create a new directory. The appropriate TID and additional pathname are passed. The directory must not exist for it to be created.

Unprotected servers will require requesters to have create permission for the subtree containing the directory in order to create a new directory. The creator's access rights to the new directory will be determined by the containing subtree permissions.

Protected servers will require requesters to have write permission on the new directory's parent directory. The access permissions granted on a created directory will be read/write permission for the creator.

Create Directory may generate the following errors:

Error Class ERRDOS:

ERRbadpath
ERRnoaccess

Error Class ERRSRV:

ERRerror
ERRaccess
ERRinvnid
<implementation specific>

Error Class ERRHRD:

<implementation specific>

5.9. Delete Directory

>From Consumer		To Consumer	
smb_com	SMBrmdir	smb_com	SMBrmdir
smb_wct	0	smb_wct	0
smb_bcc	min = 2	smb_bcc	0
smb_buf[]	ASCII -- 04		
	dir pathname		

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

Unprotected servers will require the requester to have write permission to the subtree containing the directory to be deleted.

Protected servers will require the requester to have write permission to the target directory's parent directory.

The effect of a delete will be, to some extent, dependent on the nature of the server. Normally only the referenced directory name is deleted, the directory contents are only deleted when all the directory's names have been deleted.

In some cases a delete will cause immediate destruction of the directory contents.

Delete Directory may generate the following errors:

Error Class ERRDOS:

ERRbadpath
ERRnoaccess
ERRremcd

Error Class ERRSRV:

ERRerror
ERRaccess
ERRinvnid
<implementation specific>

Error Class ERRHRD:

<implementation specific>

5.10. Delete File

>From Consumer		To Consumer	
smb_com	SMBunlink	smb_com	SMBunlink
smb_wct	1	smb_wct	0
smb_vwv[0]	attribute	smb_bcc	0
smb_bcc	min = 2		
smb_buf[]	ASCII -- 04		
	file pathname		

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

Multiple files may be deleted in response to a single request as Delete File supports "wild cards" in the file name (last component of the pathname). "?" is the wild card for single characters, "*" or "null" will match any number of filename characters within a single part of the filename component. The filename is divided into two parts -- an eight character name and a three character extension. The name and extension are divided by a ".".

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

"*" or "null" match entire pathname parts, thus "*.abc" or ".abc" will match any file with an extension of "abc". "*.*", "*" or "null" will match all files in a directory.

The attribute field indicates the attributes that the target file(s) must have. If the attribute is zero then only normal files are deleted. If the system file or hidden attributes are specified then the delete is inclusive -- both the specified type(s) of files and normal files are deleted.

Unprotected servers will require the requester to have write permission to the subtree containing the file to be deleted.

Protected servers will require the requester to have write permission to the target file's parent directory.

The effect of a delete will be, to some extent, dependent on the nature of the server. Normally only the referenced file name is deleted, the file contents are only deleted when all the file's names have been deleted and all file handles associated with it have been destroyed (closed).

In some cases (notably MS-DOS) a delete will cause immediate destruction of the file contents and invalidation of all fids associated with the file.

Delete File may generate the following errors:

Error Class ERRDOS:

ERRbadfile
ERRnoaccess

Error Class ERRSRV:

ERRerror
ERRaccess
ERRinvnid
<implementation specific>

Error Class ERRHRD:

<implementation specific>

5.11. Rename File

>From Consumer		To Consumer	
smb_com	SMBmv	smb_com	SMBmv
smb_wct	1	smb_wct	0
smb_vwv[0]	attribute	smb_bcc	0
smb_bcc	min = 4		
smb_buf[]	ASCII -- 04		
	old file pathname		
	ASCII -- 04		
	new file pathname		

The rename file message is sent to change the name of a file. The first file pathname must exist and the second must not. Both pathnames must be relative to the tid specified in the request. Open files may be renamed.

Multiple files may be renamed in response to a single request as Rename File supports "wild cards" in the file name (last component of the pathname). The wild card matching algorithm is described in the "Delete File" description.

The attribute field indicates the attributes that the target file(s) must have. If the attribute is zero then only normal files are renamed. If the system file or hidden attributes are specified then the rename is inclusive -- both the specified type(s) of files and normal files are renamed.

Unprotected servers require the requester to have both read and create permissions to the referenced subtree.

Protected servers require the requester to have write permission to the parent directories of both the source and destination files.

Rename is guaranteed to succeed if only the last component of the file pathnames differs. Other rename requests may succeed depending on the server implementation used.

Rename may generate the following errors:

Error Class ERRDOS:

ERRbadfile
ERRnoaccess
ERRdiffdevice

Error Class ERRSRV:

ERRerror
ERRaccess
ERRinvnid
<implementation specific>

Error Class ERRHRD:

<implementation specific>

5.12. Get File Attributes

>From Consumer		To Consumer	
smb_com	SMBgetatr	smb_com	SMBgetatr
smb_wct	0	smb_wct	10
smb_bcc	min = 2	smb_vwv[0]	attribute
smb_buf[]	ASCII -- 04	smb_vwv[1]	time1 low
	file pathname	smb_vwv[2]	time1 high
		smb_vwv[3]	file size low
		smb_vwv[4]	file size high
		smb_vwv[5-9]	reserved (MBZ)
		smb_bcc	0

The get file attributes message is sent to obtain information about a file. The attribute, time1, and file size fields must contain valid values for data files. The attribute and time1 fields must contain valid values for directories.

Get File Attributes may generate the following errors:

Error Class ERRDOS:

ERRbadfile
<implementation specific>

Error Class ERRSRV:

ERRerror
ERRinvnid
<implementation specific>

Error Class ERRHRD:

<implementation specific>

5.13. Set File Attributes

>From Consumer		To Consumer	
smb_com	SMBsetatr	smb_com	SMBsetatr
smb_wct	8	smb_wct	0
smb_vwv[0]	attribute	smb_bcc	0
smb_vwv[1]	time1 low		
smb_vwv[2]	time1 high		
smb_vwv[3-7]	reserved (MBZ)		
smb_bcc	min = 2		
smb_buf[]	ASCII -- 04		
	file pathname		
smb_nul[]	ASCII -- 04		
	null string		

The set file attributes message is sent to change the information about a file. Support of all parameters is optional. A server which does not implement one of the parameters will ignore that field. If the time1 field contains zero then the file's time is not changed.

Unprotected servers require the requester to have write permission to the subtree containing the referenced file.

Protected servers will allow the owner of the file to use this command. Other legitimate users will be server dependent.

Set File Attributes may generate the following errors:

Error Class ERRDOS:

ERRbadfunc
ERRbadpath
ERRnoaccess

Error Class ERRSRV:

ERRerror
ERRinvnid
ERRaccess
<implementation specific>

Error Class ERRHRD:

<implementation specific>

5.14. Lock Record

>From Consumer		To Consumer	
smb_com	SMBlock	smb_com	SMBlock
smb_wct	5	smb_wct	0
smb_vwv[0]	file handle	smb_bcc	0
smb_vwv[1]	count low		
smb_vwv[2]	count high		
smb_vwv[3]	offset low		
smb_vwv[4]	offset high		
smb_bcc	0		

The lock record message is sent to lock the given byte range. More than one non-overlapping byte range may be locked in a given file. Locks are coercive in nature. They prevent attempts to lock, read or write the locked portion of the file. Overlapping locks are not allowed. File addresses beyond the current end of file may be locked. Such locks will not cause allocation of file space.

Locks may only be unlocked by the process (pid) that performed the lock. The ability to perform locks is not tied to any file access permission.

Lock may generate the following errors:

Error Class ERRDOS:

ERRbadfid
ERRlock
<implementation specific>

Error Class ERRSRV:

ERRerror
ERRinvdevice
ERRinvnid
<implementation specific>

Error Class ERRHRD:

<implementation specific>

5.15. Unlock Record

>From Consumer		To Consumer	
smb_com	SMBunlock	smb_com	SMBunlock
smb_wct	5	smb_wct	0
smb_vwv[0]	file handle	smb_bcc	0
smb_vwv[1]	count low		
smb_vwv[2]	count high		
smb_vwv[3]	offset low		
smb_vwv[4]	offset high		
smb_bcc	0		

The unlock record message is sent to unlock the given byte range. The byte range must be identical to that specified in a prior successful lock request, and the unlock requester (pid) must be the same as the lock holder. If an unlock references an address range that is not locked it is treated as a no-op -- no action is taken and no error is generated.

Unlock may generate the following errors:

Error Class ERRDOS:

ERRbadfid
ERRlock
<implementation specific>

Error Class ERRSRV:

ERRerror
ERRinvdevice
ERRinvnid
<implementation specific>

Error Class ERRHRD:

<implementation specific>

5.16. Create Temporary File

>From Consumer		To Consumer	
smb_com	SMBctemp	smb_com	SMBctemp
smb_wct	3	smb_wct	1
smb_vwv[0]	attribute	smb_vwv[0]	file handle
smb_vwv[1]	time low	smb.bcc	min = 2
smb_vwv[2]	time high	smb_buf[]	ASCII -- 04
smb_bcc	min = 2		new file pathname
smb_buf[]	ASCII -- 04		
	directory pathname		

The server creates a data file in the directory specified in the request message and assigns a unique name to it. The file's name is returned to the requester. The file is opened in compatibility mode with read/write access for the requester.

Unprotected servers will require requesters to have create permission for the subtree containing the file. The newly created file will be opened in compatibility mode with the access mode determined by the containing subtree permissions.

Protected servers will require requesters to have write permission on the file's parent directory. The access permissions granted on a created file will be read/write permission for the creator. The newly created or truncated file is opened in read/write/compatibility mode.

Support of the create time supplied in the request is optional.

Create Temporary File may generate the following errors.

Error Class ERRDOS:

ERRbadpath
ERRnofids
ERRnoaccess

Error Class ERRSRV:

ERRerror
ERRaccess
ERRinvnid
ERRinvdevice
<implementation specific>

Error Class ERRHRD:

<implementation specific>

5.17. Process Exit

>From Consumer		To Consumer	
smb_com	SMBexit	smb_com	SMBexit
smb_wct	0	smb_wct	0
smb_bcc	0	smb_bcc	0

This command informs the server that a consumer process has terminated. The server will close all files opened by the named process. This will automatically release all locks the process holds. Note that there is not a start process message, process-ids are assigned by the consumer.

Process Exit may generate the following errors:

Error Class ERRDOS:

none

Error Class ERRSRV:

ERRerror
ERRinvnid
<implementation specific>

Error Class ERRHRD:

<implementation specific>

5.18. Make New File

>From Consumer		To Consumer	
smb_com	SMBmknew	smb_com	SMBmknew
smb_wct	3	smb_wct	1
smb_vwv[0]	attribute	smb_vwv[0]	file handle
smb_vwv[1]	time low	smb_bcc	0
smb_vwv[2]	time high		
smb_bcc	min = 2		
smb_buf[]	ASCII -- 04 file pathname		

The make new file message is sent to create a new data file. It is functionally equivalent to the create message, except it will always fail if the file already exists.

Make New File may generate the following errors:

Error Class ERRDOS:

ERRbadpath
ERRnofids
ERRnoaccess
<implementation specific>

Error Class ERRSRV:

ERRerror
ERRaccess
ERRinvnid
<implementation specific>

Error Class ERRHRD:

<implementation specific>

5.19. Check Path

>From Consumer		To Consumer	
smb_com	SMBchkpath	smb_com	SMBchkpath
smb_wct	0	smb_wct	0
smb_bcc	min = 2	smb_bcc	0
smb_buf[]	ASCII -- 04		
	directory path		

The check path message is used to verify that a path exists and is a directory. No error is returned if the given path exists and the requester has read access to it. Consumer machines which maintain a concept of a "working directory" will find this useful to verify the validity of a "change working directory" command. Note that the servers do NOT have a concept of working directory. The consumer must always supply full pathnames (relative to the tid).

Check Path may generate the following errors:

Error Class ERRDOS:

ERRbadpath
ERRnoaccess

Error Class ERRSRV:

ERRerror
ERRaccess
ERRinvnid
<implementation specific>

Error Class ERRHRD:

<implementation specific>

5.20. Get Server Attributes

>From Consumer		To Consumer	
smb_com	SMBdskattr	smb_com	SMBdskattr
smb_wct	0	smb_wct	5
smb_bcc	0	smb_vwv[0]	# allocation units/server
		smb_vwv[1]	# blocks/allocation unit
		smb_vwv[2]	# block size (in bytes)
		smb_vwv[3]	# free allocation units
		smb_vwv[4]	reserved (media identifier code)
		smb_bcc	0

This command is used to determine the total server capacity and remaining free space. The distinction between allocation units and disk blocks allows the use of the protocol with operating systems which allocate disk space in units larger than the physical disk block.

The blocking/allocation units used in this response may be independent of the actual physical or logical blocking/allocation algorithm(s) used internally by the server. However, they must accurately reflect the amount of space on the server.

The default value for smb_vwv[4] is zero.

Get Server Attributes may generate the following errors:

Error Class ERRDOS:

<implementation specific>

Error Class ERRSRV:

ERRerror
ERRinvnid
<implementation specific>

Error Class ERRHRD:

<implementation specific>

5.21. Negotiate Protocol

>From Consumer		To Consumer	
smb_com	SMBnegprot	smb_com	SMBnegprot
smb_wct	0	smb_wct	1
smb_bcc	min = 2	smb_vwv[0]	index
smb_buf[]	Dialect -- 02	smb_bcc	0
	dialect0		
	.		
	.		
	Dialect -- 02		
	dialectn		

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

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

The dialect string for the protocol specified in this document is:

PC NETWORK PROGRAM 1.0

Negotiate may generate the following errors:

Error Class ERRDOS:

<implementation specific>

Error Class ERRSRV:

ERRerror
<implementation specific>

Error Class ERRHRD:

<implementation specific>

5.22. File Search

>From Consumer		To Consumer	
smb_com	SMBsearch	smb_com	SMBsearch
smb_wct	2	smb_wct	1
smb_vwv[0]	max-count	smb_vwv[0]	count-returned
smb_vwv[1]	attribute	smb_bcc	min = 3
smb_bcc	min = 5	smb_buf[]	Variable block -- 05
smb_buf[]	ASCII -- 04		length of data
	file pathname		directory entries
	Variable block -- 05		
	length of data		
	search status		

This command is used to search directories. The file path name in the request specifies the file to be sought. The attribute field indicates the attributes that the file must have. If the attribute is zero then only normal files are returned. If the system file, hidden or directory attributes are specified then the search is inclusive -- both the specified type(s) of files and normal files are returned. If the volume label attribute is specified then the search is exclusive, and only the volume label entry is returned

The max-count field specifies the number of directory entries to be returned. The response will contain one or more directory entries as determined by the count-returned field. No more than max-count entries will be returned. Only entries that match the sought filename/attribute will be returned.

The search-status field must be null (length = 0) on the initial search request. Subsequent search requests intended to continue a search must contain the search-status field extracted from the last directory entry of the previous response. The search-status field is self-contained, for on calls containing a search-status neither the attribute or pathname fields will be valid in the request. Search-status has the following format:

```
BYTE    sr_res;         /* reserved:
                          bit    7 - reserved for consumer use
                          bit  5,6 - reserved for system use (must be preserved)
                          bits 0-4 - reserved for server (must be preserved) */
BYTE    sr_name[11];    /* pathname sought.  Format:
                          0-3 character extension, left justified (in last 3 chars) */
BYTE    sr_server[5];   /* available for server use (1st byte must be non-zero) */
BYTE    sr_res[4];      /* reserved for consumer use */
```

A File Search request will terminate when either the requested maximum number of entries that match the named file are found, or the end of directory is reached without the maximum number of matches being found. A response containing no entries indicates that no matching entries were found between the starting point of the search and the end of directory.

There may be multiple matching entries in response to a single request as File Search supports "wild cards" in the file name (last component of the pathname). The wild card matching algorithm is described in the "Delete File" description.

Unprotected servers require the requester to have read permission on the subtree containing the directory searched.

Protected servers require the requester to have read permission on the directory searched.

If a File Search requests more data than can be placed in a message of the max-xmit-size for the TID specified, the server will abort the virtual circuit to the consumer.

dir_info entries have the following format.

```
BYTE    find_buf_reserved[21];   /* reserved (search_status) */
BYTE    find_buf_attr;           /* attribute */
WORD    find_buf_time;           /* modification time (hhhhh mmmmmm xxxxx)
                                    where 'xxxxx' is in two second increments */
WORD    find_buf_date;           /* modification date (yyyyyyy mmmm ddddd) */
WORD    find_buf_size_l;         /* file size -- low word */
WORD    find_buf_size_h;         /* file size -- high word */
BYTE    find_buf_pname[13];      /* file name -- ASCII (null terminated) */
```

File Search may generate the following errors:

Error Class ERRDOS:

ERRnofiles

Error Class ERRSRV:

ERRerror
ERRaccess
ERRinvnid
<implementation specific>

Error Class ERRHRD:

<implementation specific>

5.23. Create Print File

>From Consumer		To Consumer	
smb_com	SMBsplopen	smb_com	SMBsplopen
smb_wct	2	smb_wct	1
smb_vwv[0]	length of printer setup data	smb_vwv[0]	file handle
smb_vwv[1]	mode	smb_bcc	0
smb_bcc	min = 2		
smb_buf	ASCII -- 04		
	identifier string (max 15)		

This message is sent to create a new printer file. The file handle returned can be used for subsequent write and close commands. The file name will be formed by concatenating the identifier string and a server generated number. The file will be deleted once it has been printed.

The mode field can have the following values:

0 = Text mode. (DOS servers will expand TABs.)
1 = Graphics mode.

Protected servers grant write permission to the creator of the file. No other users will be given any permissions to the file. All users will have read permission to the print queue, but only the print server has write permission to it.

Create Print File may generate the following errors:

Error Class ERRDOS:

ERRbadpath
ERRnofids
ERRnoaccess
<implementation specific>

Error Class ERRSRV:

ERRerror
ERRqfull
ERRqtoobig
ERRinvnid
<implementation specific>

Error Class ERRHRD:

<implementation specific>

5.24. Close Print File

>From Consumer		To Consumer	
smb_com	SMBsplclose	smb_com	SMBsplclose
smb_wct	1	smb_wct	0
smb_vwv[0]	file handle	smb_bcc	0
smb_bcc	0		

This message invalidates the specified file handle and queues the file for printing. The file handle must reference a print file.

Close Print File may generate the following errors:

Error Class ERRDOS:

ERRbadfid
<implementation specific>

Error Class ERRSRV:

ERRerror
ERRinvdevice
ERRqtoobig
ERRinvnid
<implementation specific>

Error Class ERRHRD:

<implementation specific>

5.25. Write Print File

>From Consumer		To Consumer	
smb_com	SMBsplwr	smb_com	SMBsplwr
smb_wct	1	smb_wct	0
smb_vwv[0]	file handle	smb_bcc	0
smb_bcc	min = 4		
smb_buf	Data block -- 01		
	length of data		
	data		

This message appends the data block to the print file specified by the file handle. The file handle must reference a print file. The first block sent to a print file must contain the printer setup data. The length of this data is specified in the Create Print File request.

If a Write Print File sends a message of length greater than the max-xmit-size for the TID specified, the server will abort the virtual circuit to the consumer.

Write Print File may generate the following errors:

Error Class ERRDOS:

ERRbadfid
ERRnoaccess
<implementation specific>

Error Class ERRSRV:

ERRerror
ERRinvdevice
ERRqtoobig
ERRinvnid
<implementation specific>

Error Class ERRHRD:

<implementation specific>

5.26. Get Print Queue

>From Consumer		To Consumer	
smb_com	SMBsplretq	smb_com	SMBsplretq
smb_wct	2	smb_wct	2
smb_vwv[0]	max_count	smb_vwv[0]	count
smb_vwv[1]	start index	smb_vwv[1]	restart index
smb_bcc	0	smb_bcc	min = 3
		smb_buf	Data block -- 01
			length of data
			queue elements

This message obtains a list of the elements currently in the print queue on the server. "start index" specifies the first entry in the queue to return, "max_count" specifies the maximum number of entries to return, this may be a positive or negative number. A positive number requests a forward search, a negative number indicates a backward search. In the response "count" indicates how many entries were actually returned. "Restart index" is the index of the entry following the last entry returned; it may be used as the start index in a subsequent request to resume the queue listing.

Get Print Queue will return less than the requested number of elements only when the top or end of the queue is encountered

The format of the queue elements returned is:

smb_date	WORD	file date (yyyyyyy mmmm ddddd)
smb_time	WORD	file time (hhhhh mmmmmm xxxxx) where 'xxxxx' is in 2 second increments
smb_status	BYTE	entry status 01 = held or stopped 02 = printing 03 = awaiting print 04 = in intercept 05 = file had error 06 = printer error 07-FF = reserved
smb_file	WORD	spool file number (from create print file request)
smb_sizelo	WORD	low word of file size
smb_sizehi	WORD	high word of file size
smb_res	BYTE	reserved
smb_name	BYTE[16]	originator name (from create print file request)

Get Print Queue may generate the following errors:

Error Class ERRDOS:

<implementation specific>

Error Class ERRSRV:

ERRerror
ERRqeof
ERRinvnid
<implementation specific>

Error Class ERRHRD:

<implementation specific>

6. Message Commands

These commands provide a message delivery system between users of systems participating in the network. The message commands cannot use VCs established for the file sharing commands. A separate VC, dedicated to messaging, must be established.

Messaging services should support message forwarding. By convention user names used for message delivery have a suffix (in byte 16) of "03", forwarded names have a suffix of "05". The algorithm for sending messages is to first attempt to deliver the message to the forwarded name, and only if this fails to attempt to deliver to the normal name.

6.1. Send Single Block Message

>From Consumer		To Consumer	
smb_com	SMBsends	smb_com	SMBsends
smb_wct	0	smb_wct	0
smb_bcc	min = 7	smb_bcc	0
smb_buf[]	ASCII -- 04		
	originator name (max 15 bytes)		
	ASCII -- 04		
	destination name (max 15 bytes)		
	Data Block -- 01		
	length of message (max 128)		
	message (max 128 bytes)		

Send Single Block Message sends a short message (up to 128 bytes in length) to a single destination (user).

The names specified in this message do not include the one byte suffix ("03" or "05").

Send Single Block Message may generate the following errors.

Error Class ERRDOS:

<implementation specific>

Error Class ERRSRV:

ERRerror
ERRinvnid
ERRpaused
ERRmsgoff
ERRnoroom
<implementation specific>

Error Class ERRHRD:

<implementation specific>

6.2. Send Broadcast Message

>From Consumer		To Consumer
smb_com	SMBsendb	No Response
smb_wct	0	
smb_bcc	min = 8	
smb_buf[]	ASCII -- 04	
	originator name (max 15 bytes)	
	ASCII -- 04	
	"*"	
	Data Block -- 01	
	length of message (max 128)	
	message (max 128 bytes)	

Send Broadcast Message sends a short message (up to 128 bytes in length) to every user in the network.

The name specified in this message does not include the one byte suffix ("03").

There is no response message to this command, thus Send Broadcast Message cannot generate errors.

6.3. Send Start of Multi-block Message

>From Consumer		To Consumer	
smb_com	SMBsendstrt	smb_com	SMBsendstrt
smb_wct	0	smb_wct	1
smb_bcc	min = 0	smb_vwv	message group ID
smb_buf[]	ASCII -- 04	smb_bcc	0
	originator name (max 15 bytes)		
	ASCII -- 04		
	destination name (max 15 bytes)		

This command informs the server that a multi-block message will be sent. The server returns a message group ID to be used to identify the message blocks when they are sent.

The names specified in this message do not include the one byte suffix ("03" or "05").

Send Start of Multi-block Message may generate the following errors.

Error Class ERRDOS:

<implementation specific>

Error Class ERRSRV:

ERRerror
ERRinvnid
ERRpaused
ERRmsgoff
ERRnoroom
<implementation specific>

Error Class ERRHRD:

<implementation specific>

6.4. Send Text of Multi-block Message

>From Consumer		To Consumer	
smb_com	SMBsendtxt	smb_com	SMBsendtxt
smb_wct	1	smb_wct	0
smb_vwv	message group ID	smb_bcc	0
smb_bcc	min = 3		
smb_buf[]	Data Block -- 01		
	length of message (max 128)		
	message (max 128 bytes)		

This command delivers a segment of a multi-block message to the server. It must contain a valid message group ID returned by an earlier Start Multi-block Message command.

A maximum of 128 bytes of message may be sent with this command. A multi-block message cannot exceed 1600 bytes in total length (sum of all segments sent with a given message group ID).

Send Text of Multi-block Message may generate the following errors.

Error Class ERRDOS:

<implementation specific>

Error Class ERRSRV:

ERRerror
ERRinvnid
ERRpaused
ERRmsgoff
ERRnoroom
<implementation specific>

Error Class ERRHRD:

<implementation specific>

6.5. Send End of Multi-block Message

>From Consumer		To Consumer	
smb_com	SMBsendend	smb_com	SMBsendend
smb_wct	0	smb_wct	0
smb_vwv	message group ID	smb_bcc	0
smb_bcc	0		

This command signals the completion of the multi-block message identified by the message group ID.

Send End of Multi-block Message may generate the following errors.

Error Class ERRDOS:

<implementation specific>

Error Class ERRSRV:

ERRerror
ERRinvnid
ERRpaused
ERRmsgoff
<implementation specific>

Error Class ERRHRD:

<implementation specific>

6.6. Forward User Name

>From Consumer		To Consumer	
smb_com	SMBfwdname	smb_com	SMBfwdname
smb_wct	0	smb_wct	0
smb_bcc	min = 2	smb_bcc	0
smb_buf[]	ASCII -- 04		
	forwarded name (max 15 bytes)		

This command informs the server that it should accept messages sent to the forwarded name.

The name specified in this message does not include the one byte suffix ("03" or "05").

Forward User Name may generate the following errors.

Error Class ERRDOS:

<implementation specific>

Error Class ERRSRV:

ERRerror
ERRinvnid
ERRrmuns
<implementation specific>

Error Class ERRHRD:

<implementation specific>

6.7. Cancel Forward

>From Consumer		To Consumer	
smb_com	SMBcancelf	smb_com	SMBcancelf
smb_wct	0	smb_wct	0
smb_bcc	min = 2	smb_bcc	0
smb_buf[]	ASCII -- 04		
	forwarded name (max 15 bytes)		

The Cancel Forward command cancels the effect of a prior Forward User Name command. The addressed server will no longer accept messages for the designated user name.

The name specified in this message does not include the one byte suffix ("05").

Cancel Forward may generate the following errors.

Error Class ERRDOS:

<implementation specific>

Error Class ERRSRV:

ERRerror
ERRinvnid
<implementation specific>

Error Class ERRHRD:

<implementation specific>

6.8. Get Machine Name

>From Consumer		To Consumer	
smb_com	SMBgetmac	smb_com	SMBgetmac
smb_wct	0	smb_wct	0
smb_bcc	0	smb_bcc	min = 2
		smb_buf[]	ASCII -- 04
			machine name (max 15 bytes)

The Get Machine Name command obtains the machine name of the target machine. It is used prior to the Cancel Forward command to determine which machine to send the Cancel Forward command to. Get Machine Name is sent to the forwarded name to be canceled, and the server then returns the machine name to which the Cancel Forward command must be sent.

Get Machine Name may return the following errors.

Error Class ERRDOS:

<implementation specific>

Error Class ERRSRV:

ERRerror
ERRinvnid
<implementation specific>

Error Class ERRHRD:

<implementation specific>

7. Data Definitions

7.1. Message Objects

attribute:The attributes of the file. Portions of this field indicate the type of file. The rest of the contents are server specific. The MS-DOS server will return the following values in attribute (bit0 is the low order bit):

Generic Attributes:
bit4 - directory

MS-DOS Attributes:
bit0 - read only file
bit1 - "hidden" file
bit2 - system file
bit3 - volume id
bit5 - archive file
bits6-15 - reserved

Support of the Generic Attributes is mandatory; support of the MS-DOS Attributes is optional. If the MS-DOS Attributes are not supported, attempts to set them must be rejected and atempts to match on them (e.g., File Search) must result in a null response.

count of bytes:The count of bytes (1 to the maximum size) read/written. The maximum size is server specific.

count left:The count of bytes not yet read/written. This field is advisory only and is used for read-ahead in the server.

count-returned:The actual number of directory entries that are returned by a file-search response.

data read/written:The actual data.

dialect-0-dialect-n:A list of dialects, each of which identifies a requested protocol and version in a string. Examples: "SNA-REV2" "TEST PROTOCOL" "RING.2"

dir_info:A data block containing an array of directory entries returned by file search.

dir pathname:An ASCII string, null terminated, that defines the location of a file within the tree. Use the '\' character to separate components. The last component names a directory. The maximum size of this field is server specific. The pathname is relative to a TID and may or may not commence with a '\'

file handle:The file identifier obtained form an open, create, make new file, and make temp file. File handles are unique within a process id.

file pathname:An ASCII string, null terminated, that defines the location of a file within the tree. Use the '\' character to separate components. The last component names a file. The maximum size of this field is server specific. The pathname is relative to a TID and may or may not commence with a '\'

file size low/hi:Low and hi words of a 32-bit long field that represents the Data file size.

identifier string:The "originator name" of the owner of a print file. The server will add a number to it to generate a unique file name. This is a null terminated ASCII string.

max-count:The maximum number of directory entries that can be returned by a file-search response.

max xmit size:The maximum size message that a server can handle.

message group IDA message group ID uniquely identifies a multi-block message.

non-owner access:The access rights of other than the owner.

offset low/hi:The low and hi words of a 32-bit offset.

owner access:The access rights of the owner.

owner id:The user id of the owner of the file.

password:May be used with the pathname for authentication by the NET USE command. This is a null terminated ASCII string.

r/w/share:This field defines the file mode. It contains fields that represent the following:

Access modes:
- Read
- Write
- Read/Write

Sharing modes:
- Exclusive
- No restriction
- Multiple Readers
- Multiple Writers

search-status:A variable block reserved for server specific information that is passed from each file search response message to the next file search request.

time1 low/hi:File modification time. these two words define a 32 bit field that contains the modification time expressed as seconds past Jan 1 1970 (local time zone). A value of zero indicates a null time field.

7.2. Data Buffer Formats (smb_buf)

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

Data Identifier Bytes		
Name	Description	Value
Data Block	See Below	01
Dialect	Null terminated ASCII String	02
Pathname	Null terminated ASCII String	03
ASCII	Null terminated ASCII String	04
Variable block	See Below	05

When the identifier indicates a data block or variable block then the format is a word indicating the length followed by the data. ASCII strings are null terminated.

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

7.3. Command Codes

The following values have been assigned for the 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 */
```

7.4. Error Codes and Classes

ERROR CLASS CODES

SUCCESS	0	The request was successful.
ERRDOS	0x01	Error is generated by the server operating system.
ERRSRV	0x02	Error is generated by the server network file manager.
ERRHRD	0x03	Error is an hardware error (MS-DOS int 24).
ERRCMD	0xFF	Command was not in the "SMB" format. (optional)

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

SUCCESS	0	The request was successful.
BUFFERED	0x54	message has been buffered
LOGGED	0x55	message has been logged
DISPLAYED	0x56	user message displayed

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.

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. [ENOENT]
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] duplicate name errors invalid rename command write to fid open for read only read on fid open for write only attempt to open read-only file for write attempt to delete read-only file attempt to set attributes of a read only file attempt to create a file on a full server directory full attempt to delete a non-empty directory invalid file type (e.g., file commands on a 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 a non-compatibility mode 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 request attempted to remove a lock held by another process. [EDEADLOCK]
ERRfilexists	80	The file named in a Create Directory or Make New File request already exists. The error may also be generated in the Create and Rename transactions. [EEXIST]

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 is invalid.
ERRbadtype	3	reserved
ERRaccess	4	The requester does not have the necessary access rights within the specified TID context for the requested function. [EACCES]
ERRinvnid	5	The tree ID (tid) specified in a command was invalid.
ERRinvnetname	6	Invalid name supplied with 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.
ERRpaused	81	Server is paused.
ERRmsgoff	82	Not receiving messages.
ERRnoroom	83	No room to buffer message.
ERRrmuns	87	Too many remote user names.
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	Invalid disk 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 compatibility mode open conflicts with an existing open on the file. [ETXTBSY]

8. Exception Handling

Exception handling is built upon the various environments supported by the file sharing protocol (see ARCHITECTURAL MODEL section). When any environment is dissolved (in either an orderly or disorderly fashion) all contained environments are dissolved. The hierarchy of environments is summarized below:

Virtual Circuit
- TID
- PID
- FID

As can be seen from this summary, the Virtual Circuit (VC) is the key environment. When a VC is dissolved the server processes (or equivalent) are terminated; the TIDs, PIDs and FIDs are invalidated, and any outstanding request is dropped -- a response will not be generated.

The termination of a PID will close all FIDs it contains. The destruction of TIDs and FIDs has no affect on other environments.

If the server receives a message with a bad format, e.g., lacks the "FFSMB" header, it may abort the VC.

If a server is unable to deliver responses to a consumer within n seconds, it considers the consumer dead and drops the VC to it (we anticipate that n will be a function of the transport round trip delay time).

Appendix A - An Example

In this example a MS-DOS machine will access a file on a remote machine that is running a server that supports MS-DOS file sharing.

STEP 1: Using protocols described elsewhere, the MS-DOS machine has obtained a virtual circuit (VC) to the server on the remote machine. The MS-DOS machine will then generate "Negotiate Message" on the VC with a dialect field that contains "PC NETWORK PROGRAM 1.0". The remote server will respond with a "Negotiate Reply Message" which will contain the index of the dialect string that contained "PC NETWORK PROGRAM 1.0", in this case 1, which indicates that it will service that protocol.

STEP 2: The MS-DOS machine now generates a "Tree Connect Message" with a pathname and a password. The remote server will respond with a "Tree Connect Response Message" indicating that the password has been validated permitting access to the associated sub-tree. A "Tree ID" is returned for future use.

STEP 3: The MS-DOS machine wishes to open and read a file on the remote server. This would be in response to a program that referenced a file on that remote system. The MS-DOS machine will generate, in response to a user program open, an "Open Message" with the "file path" of the file to be opened along with the mode information and the tree id. The file-path must not contain the path specified in the tree connect message. The server will respond with a "Open Reply Message" which will contain a file handle for use with future messages. It will also return the file size and modification time.

STEP 4: The MS-DOS machine now reads the file, in response to user program file reads. It will generate a "Read Message" with the "file handle" obtained from the "open message". The message will contain a count of bytes to be read and an offset within the file to start reading, and possibly count indicating future requests. The server will respond with a "Read Reply Message" with the count of data read and the data.

STEP 5: Some number of "Read Messages" and possibly "Write Messages" are transmitted, and eventually the file is closed by the user process. The MS-DOS machine will generate a "Close Message" which contains the "file handle" obtained from the "Open Response Message" and a new modification time. The server responds with a "Close Response Message".

STEP 6: At some time the MS-DOS machine generates a "Tree Disconnect Message" and receives a "Tree Disconnect Response Message." At this point the VC may be de-allocated.

