

**Portable Systems Group**

**NT LAN Manager SMB File Sharing Protocol Extensions**

**August 24, 1992**

*Microsoft Corporation*

# NT LAN Manager SMB File Sharing Protocol Extensions

1. Overview	1
1.1 Related Documents	1
2. Requirements	2
3. Differences from Existing Protocols	2
3.1 Negotiation of Additional Capabilities	2
3.1.1 NT Status Codes	2
3.1.2 Unicode Strings	3
3.1.3 Large File Support	3
3.1.4 Multiple-Reader Opportunistic Locks	3
3.2 NT Transact SMB	3
3.3 Extended Create and Open Semantics	3
3.4 Handle-Based Delete and Rename	4
3.5 Extended I/O Control Functions	4
3.6 Directory Change Notification	4
3.7 Extended File Information	4
3.8 Command Cancellation	4
3.9 Querying and Setting Security Descriptors on Files	4
4. Protocol Messages	4
4.1 Type Definitions	4
4.2 Symbolic Constants	7
4.3 Header Format	12
4.4 Modified SMBs	14
4.4.1 Negotiate	14
4.4.2 Session Setup And X	17
4.4.3 Find First2	20
4.4.4 Query File Information	22
4.4.5 Set File Information	25
4.4.6 Query Path Information	26
4.4.7 Set Path Information	27
4.4.8 Query File System Information	28
4.4.9 Read And X	29
4.4.10 Write And X	30
4.4.11 Read Block Raw	31
4.4.12 Write Block Raw	32
4.4.13 Locking And X	33
4.5 New SMBs	36
4.5.1 NT Transact	36
4.5.2 NT Create And X	41
4.5.3 NT Create With Security Descriptor Or EAs	49
4.5.4 NT I/O Control	55
4.5.5 NT Notify Directory Change	56
4.5.6 NT Cancel	59
4.5.7 NT Query Security Descriptor	60
4.5.8 NT Set Security Descriptor	62
4.5.9 NT Rename	63
5. Outstanding Issues	64
5.1 Network Code Page	64
5.2 FileLinkInformation	64
5.3 Flags2 vs. NT Create CreateOptions	64
5.4 Querying EAs	64
6. Revision History	65

## 1. Overview

This document describes NT LAN Manager extensions to the SMB File Sharing Protocol, the protocol that LAN Manager clients (redirectors) and servers use to communicate. (SMB stands for Server Message Block. The full term is very rarely used.) The extensions allow NT I/O system semantics to be expressed between NT systems.

This document is not intended to provide a full description of the various levels of SMB protocol that have been defined, although a single document that did so would certainly be useful. It assumes a working knowledge of the existing SMB protocol, and describes where the NT SMB protocol differs from that protocol.

### 1.1 Related Documents

The following documents describe the original SMB protocol and subsequent extensions to the protocol:

- o *Microsoft Networks/OpenNET File Sharing Protocol, Document Version 1.9, April 21, 1987; Microsoft Corporation, Intel Corporation* —the MS-NET or "core" protocol.
- o *OpenNET/Microsoft Networks File Sharing Protocol Extensions, Document Version 1.9, September 5, 1986; Intel Corporation, Microsoft Corporation* —the core protocol, adapted for Unix.
- o *Microsoft Networks SMB File Sharing Protocol Extensions Version 2.0, Document Version 3.3, November 7, 1988; Microsoft Corporation* —extensions for LAN Manager 1.0.
- o *Microsoft Networks SMB File Sharing Protocol Extensions Version 3.0, Document Version 1.11, June 19, 1990; Microsoft Corporation* —extensions for LAN Manager 2.0.
- o *Microsoft Networks SMB File Sharing Protocol Extensions, Document Version 3.4; Microsoft Corporation* —extensions for LAN Manager 2.1.

The following documents describe relevant portions of the NT operating system:

- o *NT I/O System Specification.*
- o *NT Named Pipe Specification.*
- o *NT Mailslot Specification.*
- o *NT Status Code Specification*
- o *NT Opportunistic Locking Design Note*
- o *Distributed System Architecture, Object Security Architecture*
- o *Distributed System Architecture, Security IDentifier Architecture Specification*
- o *Distributed System Architecture, Access Control List Architecture Specification*

- o *Distributed System Architecture, Subject Security Context Specification*

## 2. Requirements

The NT SMB protocol extensions must meet the following requirements:

- o The protocol must allow expression of NT I/O system semantics between NT systems. Note the absence of the word "full" in the previous sentence: the protocol implements only a subset of the full NT I/O system semantics.
- o The protocol must address internationalization (NLS) concerns. (Basically, this means that the protocol will pass Unicode strings.)
- o The protocol must support 64-bit file offsets.
- o The changes to the existing protocol must be minimized, in order to reduce the implementation time.
- o Subsets of the protocol must be available and usable from DOS clients. For example, a DOS client must be able to use Unicode without implementing an entirely new protocol.

The following is *not* a requirement on the NT SMB protocol:

- o The protocol need not address NT security issues. NT security will be addressed in a subsequent extension to the protocol.

## 3. Differences from Existing Protocols

This section describes, at a high level, the differences between the NT SMB protocol and previous versions of the SMB protocol.

### 3.1 Negotiation of Additional Capabilities

When a virtual circuit is established between a client and a server, a negotiation of protocol level and capabilities takes place. When the NT SMB protocol is negotiated, additional, optional capabilities can be negotiated. These capabilities are described in the following subsections.

#### 3.1.1 NT Status Codes

When passing of NT status codes is negotiated, the one-byte error class field and the two-byte error code field in the standard SMB header are replaced by a four-byte status field. The extra space for this status field was obtained by using reserved space in the header. NT status values are returned in this field. These status values have an architected format that incorporates the information included in the separate error class and error code fields. The *NT Status Code Specification* describes this format. When passing of NT status codes is negotiated, the *SMB\_FLAGS2\_NT\_STATUS* flag should be set in the *Flags2* field of every SMB.

### 3.1.2 Unicode Strings

When Unicode is negotiated, most strings are passed in Unicode format. This includes file names, resource names, and user names. This applies to null-terminated strings, length specified strings and the type-prefixed strings found in old SMBs.

In all cases where a string is passed in Unicode format, the Unicode string must be word-aligned with respect to the beginning of the SMB. Should the string not naturally fall on a two-byte boundary, a null byte of padding will be inserted, and the Unicode string will begin at the next address.

For type-prefixed Unicode strings, the padding byte is found after the type byte. The type byte is 0x04 (indicating SMB\_FORMAT\_ASCII) independent of whether the string is Ascii or Unicode.

For strings whose start addresses are found using offsets within the fixed part of the SMB (as opposed to simply being found at the byte following the preceding field,) it is guaranteed that the offset will be properly aligned.

Strings that are never passed in Unicode are:

- o The protocol strings in the *Negotiate* SMB request.
- o The service name string in the *Tree Connect And X* SMB.

When Unicode is negotiated, the *SMB\_FLAGS2\_UNICODE* flag should be set in the *Flags2* field of every SMB.

### 3.1.3 Large File Support

When large file support is negotiated, certain existing SMBs are modified to allow 64-bit file offsets to be passed. The SMBs affected are *Read And X*, *Write And X*, *Read Block Raw*, *Write Block Raw*, *Locking And X*, the create/open SMBs, and the query/set directory/file SMBs.

### 3.1.4 Multiple-Reader Opportunistic Locks

When multiple-reader opportunistic locks (referred to as level II oplocks) are negotiated, multiple readers of the same file can buffer read data; it is only when a write to the file is attempted that the oplock is fully broken. The *NT Opportunistic Locking Design Note* describes this feature in detail.

### 3.2 NT Transact SMB

The *NT Transact* SMB is identical to the existing *Transact* SMB, except that it allows data blocks of greater than 64KB to be transferred. This SMB is used to transfer the *NT Create with SD or EAs*, *NT Set* and *NT Query Security Descriptor*, *NT I/O Control* and *NT Rename* commands.

### 3.3 Extended Create and Open Semantics

The NT SMB protocol supports extended semantics on Create and Open by implementing two new commands: the *NT Create And X* command and the *NT Transact* function *NT Create With SD Or EAs*. These SMBs allow, for example, specification of more types of desired access and share access; opening of directories; and specification of additional options.

### 3.4 Handle-Based Delete and Rename

The NT SMB protocol allows open files to be deleted or renamed, as is done in the NT I/O system. This is accomplished by adding information levels/classes to the *Set File Information* SMB.

### 3.5 Extended I/O Control Functions

The protocol allows NT device and file system control functions to be passed transparently between NT clients and NT servers. These functions are transferred using the new *NT Transact* SMB.

### 3.6 Directory Change Notification

The *NT Directory Change Notification* SMB allows remoting of the *NtNotifyDirectoryChangeFile* service.

### 3.7 Extended File Information

The protocol supports extended file information classes in directory and file queries. This allows more accurate and complete information to be returned from an NT server to an NT client.

### 3.8 Command Cancellation

The NT SMB protocol allows active commands to be canceled. This is necessary in order to support the *NtCancelIoFile* service. An example of where this capability might be used is to cancel a pending lock request.

### 3.9 Querying and Setting Security Descriptors on Files

The NT SMB protocol supports querying and changing the security descriptor on a file. This is accomplished using the *NT Transact* functions *NT Query Security Descriptor* and *NT Set Security Descriptor* respectively.

## 4. Protocol Messages

This section describes the messages sent between the client and the server, when the NT SMB protocol is negotiated.

### 4.1 Type Definitions

The structure definitions presented in this specification are in the C language format. Where necessary for clarity and ease of presentation, the C syntax rules have been loosened.

The following basic type definitions are used in this specification:

```

typedef signed char CHAR;
typedef unsigned char UCHAR;
typedef signed short SHORT;
typedef unsigned short USHORT;
typedef signed long LONG;
typedef unsigned long ULONG;

typedef UCHAR ASCII;
typedef UCHAR ASCIIZ;

```

**ASCII** indicates a string of eight-bit characters whose length is specified externally. **ASCIIZ** indicates a string of eight-bit characters in which the end of the string is indicated by a byte equal to zero.

```

typedef SHORT WCHAR;
typedef WCHAR UNICODE;
typedef WCHAR UNICODEZ;

```

**WCHAR** denotes a single 16-bit character in Unicode format. **UNICODE** is a string of Unicode characters whose length (in bytes, not characters) is specified externally. **UNICODEZ** is a string of Unicode character in which the end of the string is indicated by a 16-bit character equal to zero.

```

typedef UCHAR ASCII_OR_UNICODE;

```

**ASCII\_OR\_UNICODE** denotes a string of ASCII or UNICODE characters. Which type of character is present is determined by negotiation between the client and the server.

```

typedef UCHAR BOOLEAN;

```

**BOOLEAN** indicates a one-byte field in which a value of zero means *FALSE* and a value of one means *TRUE*. All other values are invalid.

```

typedef struct {
    ULONG LowPart;
    LONG HighPart;
} LARGE_INTEGER;

```

**LARGE\_INTEGER** indicates a signed 64-bit integer.

```

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

```

**TIME** indicates a signed 64-bit integer representing either an absolute time or a time interval. Times are specified in units of 100ns. A positive value expresses an absolute time, where the base time (the 64-bit integer with value 0) is the beginning of the year 1601 AD in the Gregorian calendar. A negative value expresses a time interval relative to some base time, usually the current time.

**ACCESS\_MASK** indicates a longword sized structure which can be considered simply as a ULONG containing multiple single rights ORed together. The possible rights are listed in the following section.

```
typedef ULONG SECURITY_INFORMATION;
```

**SECURITY\_INFORMATION** is used to reference the security information associated with an object.

```
typedef USHORT SECURITY_DESCRIPTOR_CONTROL;
```

**SECURITY\_DESCRIPTOR\_CONTROL** is used to specify fields containing security descriptor control flags.

```
typedef ULONG DEVICE_TYPE;
```

**DEVICE\_TYPE** is used to contain NT device type values.

## **4.2 Symbolic Constants**

The following symbolic constants are used in this specification:

```
// SMB command codes
#define SMB_COM_NT_TRANSACT                0xA0
#define SMB_COM_NT_TRANSACT_SECONDARY    0xA1
#define SMB_COM_NT_CREATE_ANDX           0xA2
#define SMB_COM_NT_CANCEL                 0xA4
#define SMB_COM_NO_ANDX_COMMAND           0xFF

// NT Transaction function codes
#define NT_TRANSACT_CREATE                 1
#define NT_TRANSACT_IOCTL                  2
#define NT_TRANSACT_SET_SECURITY_DESC     3
#define NT_TRANSACT_NOTIFY_CHANGE        4
#define NT_TRANSACT_RENAME                 5
#define NT_TRANSACT_QUERY_SECURITY_DESC   6

// SMB Header Flags2 field bits
#define SMB_FLAGS2_KNOPS_LONG_NAMES       0x0001
#define SMB_FLAGS2_KNOPS_EAS              0x0002
#define SMB_FLAGS2_IS_LONG_NAME           0x0040
#define SMB_FLAGS2_PAGING_IO              0x2000
#define SMB_FLAGS2_NT_STATUS              0x4000
#define SMB_FLAGS2_UNICODE                 0x8000

// NT extensions to file info levels
#define SMB_FIND_FILE_DIRECTORY_INFO      0x101
#define SMB_FIND_FILE_FULL_DIRECTORY_INFO 0x102
#define SMB_FIND_FILE_NAMES_INFO          0x103
#define SMB_FIND_FILE_BOTH_DIRECTORY_INFO 0x104

#define SMB_QUERY_FILE_BASIC_INFO          0x101
#define SMB_QUERY_FILE_STANDARD_INFO      0x102
#define SMB_QUERY_FILE_EA_INFO            0x103
#define SMB_QUERY_FILE_NAME_INFO          0x104
#define SMB_QUERY_FILE_ALLOCATION_INFO     0x105
#define SMB_QUERY_FILE_END_OF_FILEINFO    0x106
#define SMB_QUERY_FILE_ALL_INFO           0x107
#define SMB_QUERY_FILE_ALT_NAME_INFO      0x108
#define SMB_QUERY_FILE_STREAM_INFO        0x109

#define SMB_SET_FILE_BASIC_INFO            0x101
#define SMB_SET_FILE_DISPOSITION_INFO     0x102
#define SMB_SET_FILE_ALLOCATION_INFO       0x103
#define SMB_SET_FILE_END_OF_FILE_INFO     0x104

#define SMB_QUERY_FS_LABEL_INFO            0x101
#define SMB_QUERY_FS_VOLUME_INFO          0x102
#define SMB_QUERY_FS_SIZE_INFO            0x103
#define SMB_QUERY_FS_DEVICE_INFO          0x104
#define SMB_QUERY_FS_ATTRIBUTE_INFO       0x105

// Server/workstation capabilities
```

```
#define CAP_RAW_MODE 0x0001
#define CAP_MPX_MODE 0x0002
#define CAP_UNICODE 0x0004
#define CAP_LARGE_FILES 0x0008
#define CAP_NT_SMB5 0x0010
#define CAP_RPC_REMOTE_APIS 0x0020
#define CAP_NT_STATUS 0x0040
#define CAP_LEVEL_II_OPLOCKS 0x0080
#define CAP_LOCK_AND_READ 0x0100

// Negotiate response SecurityMode field bits
#define NEGOTIATE_USER_SECURITY 0x01
#define NEGOTIATE_ENCRYPT_PASSWORDS 0x02

// (NT) Locking AndX request LockType field bits
#define LOCKING_ANDX_SHARED_LOCK 0x01
#define LOCKING_ANDX_OPLOCK_RELEASE 0x02
#define LOCKING_ANDX_CHANGE_LOCKTYPE 0x04
#define LOCKING_ANDX_CANCEL_LOCK 0x08
#define LOCKING_ANDX_LARGE_FILES 0x10

// (NT) Locking AndX request OplockLevel field values
#define OPLOCK_BROKEN_TO_NONE 0
#define OPLOCK_BROKEN_TO_II 1

// NT Create Flags bits
#define NT_CREATE_REQUEST_OPLOCK 0x02
#define NT_CREATE_REQUEST_OPBATCH 0x04
#define NT_CREATE_OPEN_TARGET_DIR 0x08

// NT Create DesiredAccess (ACCESS_MASK) field bits
#define DELETE 0x00010000
#define READ_CONTROL 0x00020000
#define WRITE_DAC 0x00040000
#define WRITE_OWNER 0x00080000

// NT Create DesiredAccess (ACCESS_MASK) field bits
#define FILE_READ_DATA 0x0001
#define FILE_LIST_DIRECTORY 0x0001
#define FILE_WRITE_DATA 0x0002
#define FILE_APPEND_DATA 0x0004
#define FILE_READ_EA 0x0008
#define FILE_WRITE_EA 0x0010
#define FILE_EXECUTE 0x0020
#define FILE_TRAVERSE 0x0020
#define FILE_READ_ATTRIBUTES 0x0080
#define FILE_WRITE_ATTRIBUTES 0x0100

// file attribute bits
#define FILE_ATTRIBUTE_READONLY 0x01
#define FILE_ATTRIBUTE_HIDDEN 0x02
```

```
#define FILE_ATTRIBUTE_SYSTEM                0x04
#define FILE_ATTRIBUTE_VOLUME                0x08
#define FILE_ATTRIBUTE_DIRECTORY            0x10
#define FILE_ATTRIBUTE_ARCHIVE              0x20
#define FILE_ATTRIBUTE_NORMAL               0x80

// file and directory share access rights
#define FILE_SHARE_READ                      0x01
#define FILE_SHARE_WRITE                    0x02
#define FILE_SHARE_DELETE                   0x04

// NT Create CreateDisposition values
#define FILE_SUPERSEDE                       0
#define FILE_OPEN                           1
#define FILE_CREATE                          2
#define FILE_OPEN_IF                        3
#define FILE_OVERWRITE                       4
#define FILE_OVERWRITE_IF                   5

// NT Create CreateOptions bits
#define FILE_DIRECTORY_FILE                  0x00000001
#define FILE_WRITE_THROUGH                  0x00000002
#define FILE_SEQUENTIAL_ONLY                0x00000004
#define FILE_NON_DIRECTORY_FILE             0x00000040
#define FILE_NO_EA_KNOWLEDGE                0x00000200
#define FILE_EIGHT_DOT_THREE_ONLY          0x00000400
#define FILE_RANDOM_ACCESS                  0x00000800
#define FILE_DELETE_ON_CLOSE                0x00001000

// NT Create SecurityFlags bits
#define SMB_SECURITY_DYNAMIC_TRACKING        0x01
#define SMB_SECURITY_EFFECTIVE_ONLY         0x02

// NT Create CreateAction return values
#define FILE_SUPERSEDED                      0
#define FILE_OPENED                          1
#define FILE_CREATED                         2
#define FILE_OVERWRITTEN                     3
#define FILE_EXISTS                          4
#define FILE_DOES_NOT_EXIST                  5

// NT Create response FileType values
typedef enum {
    FileTypeDisk                            = 0,
    FileTypeByteModePipe                     = 1,
    FileTypeMessageModePipe                  = 2,
    FileTypePrinter                          = 3,
    FileTypeCommDevice                       = 4,
    FileTypeUnknown                          = 0xFFFF
} FILE_TYPE;
```

```
// NT Create response DeviceState bits
#define DeviceStateBlocking          0x8000
#define DeviceStateEndPoint          0x4000
#define DeviceStatePipeType          0x0C00
#define DeviceStateReadMode          0x0300
#define DeviceStateIcount            0x00FF

// NT Notify Directory Change CompletionFilter bits
#define FILE_NOTIFY_CHANGE_FILE_NAME 0x00000001
#define FILE_NOTIFY_CHANGE_DIR_NAME  0x00000002
#define FILE_NOTIFY_CHANGE_ATTRIBUTES 0x00000004
#define FILE_NOTIFY_CHANGE_SIZE      0x00000008
#define FILE_NOTIFY_CHANGE_LAST_WRITE 0x00000010
#define FILE_NOTIFY_CHANGE_LAST_ACCESS 0x00000020
#define FILE_NOTIFY_CHANGE_CREATION  0x00000040
#define FILE_NOTIFY_CHANGE_EA         0x00000080
#define FILE_NOTIFY_CHANGE_SECURITY   0x00000100

// NT Notify Directory Change return Action values
#define FILE_ACTION_ADDED              0x00000001
#define FILE_ACTION_REMOVED            0x00000002
#define FILE_ACTION_MODIFIED           0x00000003
#define FILE_ACTION_RENAMED_OLD_NAME   0x00000004
#define FILE_ACTION_RENAMED_NEW_NAME   0x00000005

// NT Notify Directory Change additional status/error codes
#define STATUS_NOTIFY_ENUM_DIR         (0x0000010CL)
#define ERROR_NOTIFY_ENUM_DIR         1022

// NT RenameFlags bits
#define SMB_RENAME_REPLACE_IF_EXISTS  0x01

// NT Set/Query SD Security Information bits
#define OWNER_SECURITY_INFORMATION    (0X00000001L)
#define GROUP_SECURITY_INFORMATION    (0X00000002L)
#define DACL_SECURITY_INFORMATION     (0X00000004L)
#define SACL_SECURITY_INFORMATION     (0X00000008L)

// NT Create ImpersonationLevel values
typedef enum {
    SecurityAnonymous                = 0,
    SecurityIdentification            = 1,
    SecurityImpersonation              = 2,
    SecurityDelegation                = 3
} SECURITY_IMPERSONATION_LEVEL;
```

### 4.3 Header Format

All SMBs start with a standard header. This is the definition for the header:

```

UCHAR Protocol[4];
UCHAR Command;
union {
    struct {
        UCHAR ErrorClass;
        UCHAR Reserved;
        USHORT Error;
    } DosError;
    ULONG NtStatus;
} Status;
UCHAR Flags;
USHORT Flags2;
USHORT Reserved2[6];
USHORT Tid;
USHORT Pid;
USHORT Uid;
USHORT Mid;

```

*Protocol* —Specifies a "signature" indicating that the message is in the SMB protocol format. The signature value is "0xffSMB".

*Command* —Specifies the command code for the first (or only) command in the SMB.

*Status* —Specifies, in a response SMB, the status of the last (or only) command in the SMB that was processed by the server. When flag *SMB\_FLAGS2\_NT\_STATUS* of *Flags2* is clear, the *DosError* part of this union is used, and *ErrorClass* and *Error* are as specified in previous versions of the protocol. When flag *SMB\_FLAGS2\_NT\_STATUS* of *Flags2* is set, *NtStatus* is valid, and is a status code in the standard NT format.

*Flags* —Specifies flags. Its usage is the same as in previous versions of the protocol.

*Flags2* —Specifies flags. Its usage is the same as in previous versions of the protocol, with the following additions:

*SMB\_FLAGS2\_PAGING\_IO* —When set, indicates that a read will be permitted if the client does not have read permission but does have execute permission. This flag is only useful on a read request.

*SMB\_FLAGS2\_NT\_STATUS* —When clear, specifies that the Status field is in DOS format. When set, specifies that the Status field is in NT format.

*SMB\_FLAGS2\_UNICODE* —When clear, specifies that any *ASCII\_OR\_UNICODE* strings in this message contain ASCII characters. When set, any such strings contain Unicode characters.

*Tid*, *Pid*, *Uid*, *Mid* —Specify the tree, process, user, and multiplex identifiers. Their usage is the same as in previous versions of the protocol.

In the command-specific descriptions below, the header portion of the messages is not repeated. Where necessary, specific information about header fields is given.

## 4.4 Modified SMBs

This section defines extensions to existing SMBs.

*We don't currently list SMBs that change only to allow Unicode strings (essentially all strings in all SMBs except the protocol strings in Negotiate request and the service name string in Tree Connect AndX are Unicode). There should be a list of all such changes, albeit without full descriptions. Note the details concerning Unicode strings in section 3.1.2.*

### 4.4.1 Negotiate

The *Negotiate* command is the first SMB sent on a newly established virtual circuit. It is used to negotiate a protocol level. In the response to the command, the server sends information indicating which optional capabilities it supports. The final step in the negotiation occurs when the client sends a *Session Setup And X* command.

The request format for *Negotiate* is unchanged from previous versions of the protocol. The response format is changed when the NT SMB protocol is negotiated.

The command code for *Negotiate* is 0x72. The protocol string identifying the NT SMB protocol is "NT LANMAN 1.0".

#### Request Format (not extended)

```

UCHAR WordCount;
USHORT ByteCount;
struct {
    UCHAR Format;
    UCHAR DialectName[];
} Dialects[];

```

#### Extended Response Format

```

UCHAR WordCount;
USHORT DialectIndex;
UCHAR SecurityMode;
USHORT MaxMpxCount;
USHORT MaxNumberVcs;
ULONG MaxBufferSize;
ULONG MaxRawSize;
ULONG SessionKey;
ULONG Capabilities;
TIME ServerTime;
USHORT ServerTimeZone;
UCHAR Reserved;
USHORT ByteCount;
UCHAR PasswordEncryptionKey[];

```

*WordCount* —Must contain the value 17.

*DialectIndex* —Specifies the array index (within the *Dialects* array in the request) of the dialect selected by the server. The first entry is index 0.

*SecurityMode* —Specifies the following flags:

*NEGOTIATE\_USER\_SECURITY* —When clear, indicates that the server is operating under share-level security. When set, indicates that the server is operating under user-level security. (NT servers always negotiate user-level security.)

*NEGOTIATE\_ENCRYPT\_PASSWORDS* —When clear, indicates that the client should not encrypt passwords. When set, indicates that the client should encrypt passwords. (NT servers always negotiate password encryption.)

(Note that the fields from *SecurityMode* to *Reserved* are all naturally aligned.)

*MaxMpxCount* —Specifies the maximum number of requests the server will allow the client to have active simultaneously.

*MaxNumberVcs* —Specifies the maximum number of virtual circuits the server will allow the client to have active to the server.

*MaxBufferSize* —Specifies the maximum message size the server can send or receive in a normal SMB.

*MaxRawSize* —Specifies the maximum message size the server can send or receive in a raw mode data buffer.

*SessionKey* —Specifies the unique identifier for the session assigned by the server.

*Capabilities* —Specifies the following flags:

*CAP\_RAW\_MODE* —When set, indicates that the server supports the Read Block Raw and Write Block Raw commands.

*CAP\_MPX\_MODE* —When set, indicates that the server supports the Read Block Multiplexed and Write Block Multiplexed commands.

*CAP\_UNICODE* —When set, indicates that the server recognizes Unicode strings.

*CAP\_LARGE\_FILES* —When set, indicates that the server supports large files and 64-bit file offsets.

*CAP\_NT\_SMBS* —When set, indicates that the server supports the new SMBs added in this version of the protocol.

*CAP\_RPC\_REMOTE\_APIS* —When set, indicates that the server supports remote APIs via RPC.

*CAP\_NT\_STATUS* —When set, indicates that the server recognizes NT-style status codes.

*CAP\_LEVEL\_II\_OPLOCKS* —When set, indicates that the server supports level II oplocks.

*CAP\_LOCK\_AND\_READ* —When set, indicates that the server can process the *LockAndRead* and *WriteAndUnlock* SMBs. This capability is present in older versions of the protocol as a bit in the *Flags* field in the *Negotiate* response. In the *NT Negotiate* response, the old bit is ignored.

*ServerTime* —System time of the server.

*ServerTimeZone* —Time zone of the server, expressed as minutes from UTC.

*Reserved* —Is a reserved field and should be zero.

*ByteCount* —Specifies the total length, in bytes, of *PasswordEncryptionKey*.

*PasswordEncryptionKey* —Specifies the password encryption key assigned by the server.

#### 4.4.2 Session Setup And X

The *Session Setup And X* command is the second command sent by the client on a newly established virtual circuit. Although multiple sessions may be set up on a circuit, the first *Session Setup And X* command carries the final negotiated parameters for the circuit.

In the request, there are now two password encodings and their lengths, as well as a new *Capabilities* field. In the response, there is a new *PrimaryDomain* field.

The command code for *Session Setup And X* is 0x73.

##### Extended Request Format

```

UCHAR WordCount;
UCHAR AndXCommand;
UCHAR AndXReserved;
USHORT AndXOffset;
USHORT MaxBufferSize;
USHORT MaxMpxCount;
USHORT VcNumber;
ULONG SessionKey;
USHORT CaseInsensitivePasswordLength;
USHORT CaseSensitivePasswordLength;
ULONG Reserved;
ULONG Capabilities;
USHORT ByteCount;
UCHAR CaseInsensitivePassword[];
UCHAR CaseSensitivePassword[];
ASCII_OR_UNICODE Accountname[];
ASCII_OR_UNICODE PrimaryDomain[];
ASCII_OR_UNICODE NativeOS[];
ASCII_OR_UNICODE NativeLanMan[];

```

*WordCount* —Must contain the value 13.

*AndXCommand* —Specifies the command code for the next command in the chain, if any.

*AndXReserved* —Is a reserved field and should be zero.

*AndXOffset* —Specifies the offset to the next command in the chain.

*MaxBufferSize* —Specifies the maximum message size the client can receive in a normal SMB.

*MaxMpxCount* —Specifies the maximum number of requests the client will have active simultaneously.

*VcNumber* —Specifies the virtual circuit number assigned by the client.

*SessionKey* —Specifies the unique identifier for the session assigned by the server in the *Negotiate* response.

*CaseInsensitivePasswordLength* —Specifies the length of *CaseInsensitivePassword* in bytes.

*CaseSensitivePasswordLength* —Specifies the length of *CaseSensitivePassword* in bytes.

*Reserved* —Is a reserved field and should be zero.

*Capabilities* —Specifies the following flags:

*CAP\_NT\_STATUS* —When set, indicates that the client recognizes NT-style status codes.

*CAP\_LEVEL\_II\_OPLOCKS* —When set, indicates that the client supports level II oplocks.

*CAP\_UNICODE* —When set, indicates that the client recognizes Unicode strings.

*CAP\_LARGE\_FILES* —When set, indicates that the client supports large files and 64-bit file offsets.

*CAP\_NT\_SMBS* —When set, indicates that the client supports the new SMBs added in this version of the protocol.

*ByteCount* —Specifies the total length, in bytes, of *CaseInsensitivePassword*, *CaseSensitivePassword*, *Accountname*, *PrimaryDomain*, *NativeOS* and *NativeLanMan*.

*CaseInsensitivePassword* —Contains an encrypted representation of the case-insensitive password for the account that is being set up. The plain-text form of the password is in OEM characters.

*CaseSensitivePassword* —Contains an encrypted representation of the case-sensitive password for the account that is being set up. The plain-text form of the password is always in Unicode characters.

*Accountname* —Specifies the username for the account that is being set up. The string is NUL terminated.

*PrimaryDomain* —Specifies the logon domain of the user. This string is NUL terminated.

*NativeOS* —Specifies the client operating system. This string is NUL terminated.

*NativeLanMan* —Specifies the client LAN Manager type. This string is NUL terminated.

Extended Response Format

```
UCHAR WordCount;  
UCHAR AndXCommand;  
UCHAR AndXReserved;  
USHORT AndXOffset;  
USHORT Action;  
USHORT ByteCount;  
ASCII_OR_UNICODE NativeOS[];  
ASCII_OR_UNICODE NativeLanMan[];  
ASCII_OR_UNICODE PrimaryDomain[];
```

*WordCount* —Must contain the value 13.

*AndXCommand* —Specifies the command code for the next command in the chain, if any.

*AndXReserved* —Is a reserved field and should be zero.

*AndXOffset* —Specifies the offset to the next command in the chain.

*Action* —Request mode. Bit 0 indicates that logon was successful, but as guest.

*ByteCount* —Specifies the total length, in bytes of *NativeOS*, *NativeLanMan* and *PrimaryDomain*.

*NativeOS* —Specifies the server operating system. This string is NUL terminated.

*NativeLanMan* —Specifies the server LAN Manager type. This string is NUL terminated.

*PrimaryDomain* —Specifies the logon domain of the user. This string is NUL terminated.

### 4.4.3 Find First2

When the NT SMB protocol has been negotiated, the *Find First2 Transact2* function supports the following additional information classes:

- o *SMB\_FIND\_FILE\_DIRECTORY\_INFO* —Equivalent to *FileDirectoryInformation* in the NT I/O system. Uses the following response data format.

```
ULONG NextEntryOffset;
ULONG FileIndex;
LARGE_INTEGER CreationTime;
LARGE_INTEGER LastAccessTime;
LARGE_INTEGER LastWriteTime;
LARGE_INTEGER ChangeTime;
LARGE_INTEGER EndOfFile;
LARGE_INTEGER AllocationSize;
ULONG FileAttributes;
ULONG FileNameLength;
UNICODE FileName[];
```

- o *SMB\_FIND\_FILE\_FULL\_DIRECTORY\_INFO* —Equivalent to *FileFullDirectoryInformation* in the NT I/O system. Uses the following response data format.

```
ULONG NextEntryOffset;
ULONG FileIndex;
LARGE_INTEGER CreationTime;
LARGE_INTEGER LastAccessTime;
LARGE_INTEGER LastWriteTime;
LARGE_INTEGER ChangeTime;
LARGE_INTEGER EndOfFile;
LARGE_INTEGER AllocationSize;
ULONG FileAttributes;
ULONG FileNameLength;
ULONG EaSize;
UNICODE FileName[];
```

- o *SMB\_FIND\_FILE\_NAMES\_INFO* —Equivalent to *FileNamesInformation* in the NT I/O system. Uses the following response data format.

```
ULONG NextEntryOffset;
ULONG FileIndex;
ULONG FileNameLength;
UNICODE FileName[];
```

- o *SMB\_FIND\_FILE\_BOTH\_DIRECTORY\_INFO* —Equivalent to *FileBothDirectoryInformation* in the NT I/O system. Uses the following response data format.

```
ULONG NextEntryOffset;  
ULONG FileIndex;  
LARGE_INTEGER CreationTime;  
LARGE_INTEGER LastAccessTime;  
LARGE_INTEGER LastWriteTime;  
LARGE_INTEGER ChangeTime;  
LARGE_INTEGER EndOfFile;  
LARGE_INTEGER AllocationSize;  
ULONG FileAttributes;  
ULONG FileNameLength;  
ULONG EaSize;  
CHAR ShortNameLength;  
UNICODE ShortName[12];  
UNICODE FileName[];
```

Note that the *FileName* and *ShortName* fields in each of these additional information levels are always in Unicode format, even if Unicode is not a negotiated capability for the session.

#### 4.4.4 Query File Information

When the NT SMB protocol has been negotiated, the *Query File Information Transact2* function supports the following additional information classes:

- o *SMB\_QUERY\_FILE\_BASIC\_INFO* —Equivalent to *FileBasicInformation* in the NT I/O system. Uses the following response data format.

```
typedef struct {
    LARGE_INTEGER CreationTime;
    LARGE_INTEGER LastAccessTime;
    LARGE_INTEGER LastWriteTime;
    LARGE_INTEGER ChangeTime;
    ULONG FileAttributes;
} FILE_BASIC_INFORMATION;
```

- o *SMB\_QUERY\_FILE\_STANDARD\_INFO* —Equivalent to *FileStandardInformation* in the NT I/O system. Uses the following response data format.

```
typedef struct {
    LARGE_INTEGER AllocationSize;
    LARGE_INTEGER EndOfFile;
    ULONG NumberOfLinks;
    BOOLEAN DeletePending;
    BOOLEAN Directory;
} FILE_STANDARD_INFORMATION;
```

- o *SMB\_QUERY\_FILE\_EA\_INFO* —Equivalent to *FileEaInformation* in the NT I/O system. Uses the following response data format.

```
typedef struct {
    ULONG EaSize;
} FILE_EA_INFORMATION;
```

- o *SMB\_QUERY\_FILE\_NAME\_INFO* —Equivalent to *FileNameInformation* in the NT I/O system. Uses the following response data format.

```
typedef struct {
    ULONG FileNameLength;
    UNICODE FileName[];
} FILE_NAME_INFORMATION;
```

- o *SMB\_QUERY\_FILE\_ALLOCATION\_INFO* —Equivalent to *FileAllocationInformation* in the NT I/O system. Uses the following response data format.

```
typedef struct {
    LARGE_INTEGER AllocationSize;
} FILE_ALLOCATION_INFORMATION;
```

- o *SMB\_QUERY\_FILE\_END\_OF\_FILEINFO* —Equivalent to *FileEndOfFileInformation* in the NT I/O system. Uses the following response data format.

```
typedef struct {
    LARGE_INTEGER EndOfFile;
} FILE_END_OF_FILE_INFORMATION;
```

- o *SMB\_QUERY\_FILE\_ALL\_INFO* —Roughly equivalent to *FileAllInformation* in the NT I/O system. Uses the following response data format.

```
typedef struct {
    FILE_BASIC_INFORMATION BasicInformation;
    FILE_STANDARD_INFORMATION StandardInformation;
    FILE_INTERNAL_INFORMATION InternalInformation;
    FILE_EA_INFORMATION EaInformation;
    FILE_ACCESS_INFORMATION AccessInformation;
    FILE_POSITION_INFORMATION PositionInformation;
    FILE_MODE_INFORMATION ModeInformation;
    FILE_ALIGNMENT_INFORMATION AlignmentInformation;
    FILE_NAME_INFORMATION NameInformation;
} FILE_ALL_INFORMATION;
```

where *FILE\_BASIC\_INFORMATION*, *FILE\_STANDARD\_INFORMATION*, *FILE\_EA\_INFORMATION* and *FILE\_EA\_NAME\_INFORMATION* are as described above, and the remaining types are as follows:

```
typedef struct {
    LARGE_INTEGER IndexNumber;
} FILE_INTERNAL_INFORMATION;

typedef struct {
    ACCESS_MASK AccessFlags;
} FILE_ACCESS_INFORMATION;

typedef struct {
    LARGE_INTEGER CurrentByteOffset;
} FILE_POSITION_INFORMATION;

typedef struct {
    ULONG Mode;
} FILE_MODE_INFORMATION;

typedef struct {
    ULONG AlignmentRequirement;
} FILE_ALIGNMENT_INFORMATION;
```

- o *SMB\_QUERY\_FILE\_ALT\_NAME\_INFO* —Equivalent to *FileAlternateNameInformation* in the NT I/O system. Uses the same response data format as *SMB\_QUERY\_FILE\_NAME\_INFO* above.
- o *SMB\_QUERY\_FILE\_STREAM\_INFO* —Equivalent to *FileStreamInformation* in the NT I/O system. Uses the following response data format.

```
typedef struct {
    ULONG NextEntryOffset;
    LARGE_INTEGER StreamSize;
    LARGE_INTEGER StreamAllocationSize;
    ULONG StreamNameLength;
    UNICODE StreamName[];
} FILE_STREAM_INFORMATION;
```

Note that the *FileName* field in *FILE\_NAME\_INFORMATION* and *StreamName* field in *FILE\_STREAM\_INFORMATION* are always in Unicode format, even if Unicode is not a negotiated capability for the session.

#### 4.4.5 Set File Information

When the NT SMB protocol has been negotiated, the *Set File Information Transact2* function supports the following additional information classes:

- o *SMB\_SET\_FILE\_BASIC\_INFO* —Equivalent to *FileBasicInformation* in the NT I/O system. Uses the following request data format.

```
LARGE_INTEGER CreationTime;  
LARGE_INTEGER LastAccessTime;  
LARGE_INTEGER LastWriteTime;  
LARGE_INTEGER ChangeTime;  
ULONG FileAttributes;
```

- o *SMB\_SET\_FILE\_DISPOSITION\_INFO* —Equivalent to *FileDispositionInformation* in the NT I/O system. Uses the following request data format.

```
BOOLEAN DeleteFile;
```

- o *SMB\_SET\_FILE\_ALLOCATION\_INFO* —Equivalent to *FileAllocationInformation* in the NT I/O system. Uses the following request data format.

```
LARGE_INTEGER AllocationSize;
```

- o *SMB\_SET\_FILE\_END\_OF\_FILE\_INFO* —Equivalent to *FileEndOfFileInformation* in the NT I/O system. Uses the following request data format.

```
LARGE_INTEGER EndOfFile;
```

#### 4.4.6 Query Path Information

When the NT SMB protocol has been negotiated, the *Query Path Information Transact2* function supports the same additional information classes as *Query File Information*, namely:

- o *SMB\_QUERY\_FILE\_BASIC\_INFO*
- o *SMB\_QUERY\_FILE\_STANDARD\_INFO*
- o *SMB\_QUERY\_FILE\_EA\_INFO*
- o *SMB\_QUERY\_FILE\_NAME\_INFO*
- o *SMB\_QUERY\_FILE\_ALLOCATION\_INFO*
- o *SMB\_QUERY\_FILE\_END\_OF\_FILEINFO*
- o *SMB\_QUERY\_FILE\_ALLINFO*

See the description of *Query File Information* for the data formats.

#### 4.4.7 Set Path Information

When the NT SMB protocol has been negotiated, the *Set Path Information Transact2* function supports the same additional information classes as *Set File Information*, namely:

- o *SMB\_SET\_FILE\_BASIC\_INFO*
- o *SMB\_SET\_FILE\_DISPOSITION\_INFO*
- o *SMB\_SET\_FILE\_ALLOCATION\_INFO*
- o *SMB\_SET\_FILE\_END\_OF\_FILE\_INFO*

See the description of *Set File Information* for the data formats.

#### 4.4.8 Query File System Information

When the NT SMB protocol has been negotiated, the *Query File System Information Transact2* function supports the following additional information classes:

- o *SMB\_QUERY\_FS\_LABEL\_INFO* —Equivalent to *FileFsLabelInformation* in the NT I/O system. Uses the following response data format.

```
ULONG VolumeLabelLength;
UNICODE VolumeLabel[];
```

- o *SMB\_QUERY\_FS\_VOLUME\_INFO* —Equivalent to *FileFsVolumeInformation* in the NT I/O system. Uses the following response data format.

```
LARGE_INTEGER VolumeCreationTime;
ULONG VolumeSerialNumber;
ULONG VolumeLabelLength;
BOOLEAN SupportsObjects;
UNICODE VolumeLabel[];
```

- o *SMB\_QUERY\_FS\_SIZE\_INFO* —Equivalent to *FileFsSizeInformation* in the NT I/O system. Uses the following response data format.

```
LARGE_INTEGER TotalAllocationUnits;
LARGE_INTEGER AvailableAllocationUnits;
ULONG SectorsPerAllocationUnit;
ULONG BytesPerSector;
```

- o *SMB\_QUERY\_FS\_DEVICE\_INFO* —Equivalent to *FileFsDeviceInformation* in the NT I/O system. Uses the following response data format.

```
DEVICE_TYPE DeviceType;
ULONG Characteristics;
```

- o *SMB\_QUERY\_FS\_ATTRIBUTE\_INFO* —Equivalent to *FileFsAttributeInformation* in the NT I/O system. Uses the following response data format.

```
ULONG FileSystemAttributes;
LONG MaximumComponentNameLength;
ULONG FileSystemNameLength;
UNICODE FileSystemName[];
```

Note that the *VolumeLabel* and *FileSystemName* fields in these additional information levels are always in Unicode format, even if Unicode is not a negotiated capability for the session.

#### 4.4.9 Read And X

When the NT SMB protocol has been negotiated, the *Read And X* command supports the specification of a 64-bit file offset. This is accomplished by allowing the client to increase the *WordCount* field by two (from 10 to 12) and place the high four bytes of the file offset between the *Remaining* and *ByteCount* fields. The server uses the value of *WordCount* to determine whether a 32-bit or 64-bit offset is present. The response format is not extended.

#### 4.4.10 Write And X

When the NT SMB protocol has been negotiated, the *Write And X* command supports the specification of a 64-bit file offset. This is accomplished by allowing the client to increase the *WordCount* field by two (from 12 to 14) and place the high four bytes of the file offset between the *DataOffset* and *ByteCount* fields. The server uses the value of *WordCount* to determine whether a 32-bit or 64-bit offset is present. The response format is not extended.

#### 4.4.11 Read Block Raw

When the NT SMB protocol has been negotiated, the *Read Block Raw* command supports the specification of a 64-bit file offset. This is accomplished by allowing the client to increase the *WordCount* field by two (from 8 to 10) and place the high four bytes of the file offset between the *Reserved* and *ByteCount* fields. The server uses the value of *WordCount* to determine whether a 32-bit or 64-bit offset is present. The response format is not extended.

#### 4.4.12 Write Block Raw

When the NT SMB protocol has been negotiated, the *Write Block Raw* command supports the specification of a 64-bit file offset. This is accomplished by allowing the client to increase the *WordCount* field by two (from 12 to 14) and place the high four bytes of the file offset between the *DataOffset* and *ByteCount* fields. The server uses the value of *WordCount* to determine whether a 32-bit or 64-bit offset is present. The response format is not extended.

#### 4.4.13 Locking And X

This SMB is used to lock and unlock byte ranges. When the NT SMB protocol has been negotiated, the client may specify byte ranges using 64-bit offsets.

The *Locking And X* SMB is also used in breaking oplocks. The server sends a *Locking And X* SMB with a special bit set to indicate that an oplock is being broken, and the client sends a *Locking And X* SMB with the same bit set to release the oplock. When the NT SMB protocol has been negotiated, this SMB is extended to allow the server to specify oplock levels, by appropriating the upper byte of *LockType* for the new *OplockLevel* field.

##### Extended Request Format

```

UCHAR WordCount;
UCHAR AndXCommand;
UCHAR AndXReserved;
USHORT AndXOffset;
USHORT Fid;
UCHAR LockType;
UCHAR OplockLevel;
ULONG Timeout;
USHORT NumberOfUnlocks;
USHORT NumberOfLocks;
USHORT ByteCount;
LOCK_RANGE Unlocks[];
LOCK_RANGE Locks[];

```

*WordCount* —Must contain the value 8.

*AndXCommand* —Specifies the command code for the next command in the chain, if any.

*AndXReserved* —Is a reserved field and should be zero.

*AndXOffset* —Specifies the offset to the next command in the chain.

*Fid* —Specifies the identifier for the open file instance.

*LockType* —Specifies the following flags:

*LOCKING\_ANDX\_SHARED\_LOCK* —When clear, indicates that an exclusive lock is being requested. When set, indicates that a shared lock is being requested.

*LOCKING\_ANDX\_OPLOCK\_RELEASE* —When set in an unsolicited message from the server, indicates that the oplock previously granted to the client for this file is being broken. The *OplockLevel* field indicates the new granted oplock level. When this bit is set in a message from the client, it indicates that the client is acknowledging the oplock level change.

*LOCKING\_ANDX\_CHANGE\_LOCKTYPE* —When set, indicates that the client wishes to unlock the specified region, and relock it according to the mode specified by the

*LOCKING\_ANDX\_SHARED\_LOCK* bit, atomically. Currently, the NT LAN Manager server does not support this operation.

*LOCKING\_ANDX\_CANCEL\_LOCK* —When set, indicates that the client wishes to cancel the outstanding lock request. This is used to support cancelling of lock request for client which do not support the Cancel SMB. The cancel SMB is the preferred method for cancelling requests.

*LOCKING\_ANDX\_LARGEFILES* —When clear, indicates that the lock ranges are specified using 32-bit offsets. When set, indicates that the lock ranges are specified using 64-bit offsets.

*OplockLevel* —Indicates the new oplock level when an existing oplock is being broken. The legal values are *OPLOCK\_BROKEN\_TO\_NONE* and *OPLOCK\_BROKEN\_TO\_II*, where *OPLOCK\_BROKEN\_TO\_NONE* means that the client no longer has an oplock on the file, and *OPLOCK\_BROKEN\_TO\_II* means that the client now has a multiple-reader oplock. If the client has indicated that it does not understand level II oplocks, the server will always specify *OPLOCK\_BROKEN\_TO\_NONE* in this field when breaking an oplock.

*Timeout* —Specifies the amount of time to wait for the specified locks to be granted.

*NumberOfUnlocks* —Specifies the number of unlock ranges that are specified in *UnlockRanges*.

*NumberOfLocks* —Specifies the number of lock ranges that are specified in *LockRanges*.

*ByteCount* —Specifies the total size, in bytes, of the *UnlockRanges* and *LockRanges* fields.

*UnlockRanges* —Supplies zero or more ranges of existing locks that are to be released. When *LOCKING\_ANDX\_LARGE\_FILES* is clear, the ranges are specified in the 32-bit lock range format given below. When *LOCKING\_ANDX\_LARGE\_FILES* is set, the ranges are specified in the 64-bit lock range format given below.

*LockRanges* —Supplies zero or more ranges of locks that are to be obtained. When *LOCKING\_ANDX\_LARGE\_FILES* is clear, the ranges are specified in the 32-bit lock range format given below. When *LOCKING\_ANDX\_LARGE\_FILES* is set, the ranges are specified in the 64-bit lock range format given below.

Response Format (not extended)

```
UCHAR WordCount;  
UCHAR AndXCommand;  
UCHAR AndXReserved;  
USHORT AndXOffset;  
USHORT ByteCount;
```

32-bit Lock Range Format (not extended)

```
USHORT Pid;  
ULONG Offset;  
ULONG Length;
```

64-bit Lock Range Format (new)

```
USHORT Pid;  
USHORT Pad;  
LARGE_INTEGER Offset;  
ULONG Length;
```

## 4.5 New SMBs

This section defines SMBs that are new in this version of the protocol. Clients are allowed to send these SMBs only if the server indicates that it supports them in the *Negotiate* response.

### 4.5.1 NT Transact

The *NT Transact* SMB is used for commands that potentially need to transfer a large amount of data (greater than the negotiated buffer size). *NT Transact* operates in the same way as the original *Transact*. See *Microsoft Networks SMB File Sharing Protocol Extensions Version 2.0* for a detailed description of *Transact*.

The command code for the *NT Transact* primary SMB is *SMB\_COM\_NT\_TRANSACT*. The command code for the *NT Transact* secondary SMB is *SMB\_COM\_NT\_TRANSACT\_SECONDARY*.

The interim response format has command code *SMB\_COM\_NT\_TRANSACT* and is only ever returned in response to a primary request. The response format can have either *SMB\_COM\_NT\_TRANSACT* or *SMB\_COM\_NT\_TRANSACT\_SECONDARY* as its command code and is returned in response to either a primary or secondary request. Interim responses and primary responses are distinguished by the value of the *WordCount* field.

#### Primary Request Format

```

    UCHAR WordCount;
    UCHAR MaxSetupCount;
    USHORT Flags;
    ULONG TotalParameterCount;
    ULONG TotalDataCount;
    ULONG MaxParameterCount;
    ULONG MaxDataCount;
    ULONG ParameterCount;
    ULONG ParameterOffset;
    ULONG DataCount;
    ULONG DataOffset;
    UCHAR SetupCount;
    USHORT Function;
    USHORT Setup[];
    USHORT ByteCount;
    UCHAR Pad1[];
    UCHAR Parameters[];
    UCHAR Pad2[];
    UCHAR Data[];

```

*WordCount* —The number of words following. Must be equal to  $19 + \textit{SetupCount}$ .

*MaxSetupCount* —Specifies the maximum number of setup words that the server may return in the response.

*Flags* —Specifies flags. None are currently defined.

*TotalParameterCount* —Specifies the total number of parameter bytes that are being sent in the request.

*TotalDataCount* —Specifies the total number of data bytes that are being sent in the request.

*MaxParameterCount* —Specifies the maximum number of parameter bytes that the server may return in the response.

*MaxDataCount* —Specifies the maximum number of data bytes that the server may return in the response.

*ParameterCount* —Specifies the number of parameter bytes that are present in this SMB.

*ParameterOffset* —Specifies the offset from the start of the SMB header to the parameter bytes.

*DataCount* —Specifies the number of data bytes that are present in this SMB.

*DataOffset* —Specifies the offset from the start of the SMB header to the data bytes.

*SetupCount* —Specifies the number of setup words in the request.

*Function* —Specifies the transaction function code.

*Setup* —Supplies setup words, the contents of which are interpreted based on the function code.

*ByteCount* —The number of bytes following.

*Pad1* —Zero to three pad bytes to align *Parameters* on a four-byte boundary. The pad bytes should be zero. If *ParameterCount* is zero, *Pad1* is not present.

*Parameters* —*ParameterCount* parameter bytes.

*Pad2* —Zero to three pad bytes to align *Data* on a four-byte boundary. The pad bytes should be zero. If *DataCount* is zero, *Pad2* is not present.

*Data* —*DataCount* data bytes.

### Interim Response Format

```
UCHAR WordCount;  
USHORT ByteCount;
```

*WordCount* —Must contain the value 0.

*ByteCount* —Must contain the value 0.

## Secondary Request Format

```
UCHAR WordCount;  
UCHAR Reserved1;  
USHORT Reserved2;  
ULONG TotalParameterCount;  
ULONG TotalDataCount;  
ULONG ParameterCount;  
ULONG ParameterOffset;  
ULONG ParameterDisplacement;  
ULONG DataCount;  
ULONG DataOffset;  
ULONG DataDisplacement;  
UCHAR Reserved3;  
USHORT ByteCount;  
UCHAR Pad1[];  
UCHAR Parameters[];  
UCHAR Pad2[];  
UCHAR Data[];
```

*WordCount* —Must contain the value 18.

*Reserved1* —Is a reserved field and should be zero.

*Reserved2* —Is a reserved field and should be zero.

*TotalParameterCount* —Specifies the total number of parameter bytes that are being sent in the request.

*TotalDataCount* —Specifies the total number of data bytes that are being sent in the request.

*ParameterCount* —Specifies the number of parameter bytes that are present in this SMB.

*ParameterOffset* —Specifies the offset from the start of the SMB header to the parameter bytes.

*ParameterDisplacement* —Specifies the offset from the start of the overall parameter block to the parameter bytes that are contained in this message.

*DataCount* —Specifies the number of data bytes that are present in this SMB.

*DataOffset* —Specifies the offset from the start of the SMB header to the data bytes.

*DataDisplacement* —Specifies the offset from the start of the overall data block to the data bytes that are contained in this message.

*Reserved3* —Is a reserved field and should be zero.

*ByteCount* —The number of bytes following.

*Pad1* —Zero to three pad bytes to align *Parameters* on a four-byte boundary. The pad bytes should be zero. If *ParameterCount* is zero, *Pad1* is not present.

*Parameters* —*ParameterCount* parameter bytes.

*Pad2* —Zero to three pad bytes to align *Data* on a four-byte boundary. The pad bytes should be zero. If *DataCount* is zero, *Pad2* is not present.

*Data* —*DataCount* data bytes.

### Response Format

```

UCHAR WordCount;
UCHAR Reserved1;
USHORT Reserved2;
ULONG TotalParameterCount;
ULONG TotalDataCount;
ULONG ParameterCount;
ULONG ParameterOffset;
ULONG ParameterDisplacement;
ULONG DataCount;
ULONG DataOffset;
ULONG DataDisplacement;
UCHAR SetupCount;
USHORT Setup[];
USHORT ByteCount;
UCHAR Pad1[];
UCHAR Parameters[];
UCHAR Pad2[];
UCHAR Data[];

```

*WordCount* —Must contain the value  $18 + \textit{SetupCount}$ .

*Reserved1* —Is a reserved field and should be zero.

*Reserved2* —Is a reserved field and should be zero.

*TotalParameterCount* —Specifies the total number of parameter bytes that are being sent in the response.

*TotalDataCount* —Specifies the total number of data bytes that are being sent in the response.

*ParameterCount* —Specifies the number of parameter bytes that are present in this SMB.

*ParameterOffset* —Specifies the offset from the start of the SMB header to the parameter bytes.

*ParameterDisplacement* —Specifies the offset from the start of the overall parameter block to the parameter bytes that are contained in this message.

*DataCount* —Specifies the number of data bytes that are present in this SMB.

*DataOffset* —Specifies the offset from the start of the SMB header to the data bytes.

*DataDisplacement* —Specifies the offset from the start of the overall data block to the data bytes that are contained in this message.

*SetupCount* —Specifies the number of setup words in the response.

*Setup* —Supplies setup words, the contents of which are interpreted based on the function code.

*ByteCount* —The number of bytes following.

*Pad1* —Zero to three pad bytes to align *Parameters* on a four-byte boundary. The pad bytes should be zero. If *ParameterCount* is zero, *Pad1* is not present.

*Parameters* —*ParameterCount* parameter bytes.

*Pad2* —Zero to three pad bytes to align *Data* on a four-byte boundary. The pad bytes should be zero. If *DataCount* is zero, *Pad2* is not present.

*Data* —*DataCount* data bytes.

### 4.5.2 NT Create And X

The *NT Create And X* command is used to create or open a file or a directory. When EAs or an SD must be applied to the file, the *NT Transact* function *Create With SD Or EAs* is used.

The command code for *NT Create And X* is *SMB\_COM\_NT\_CREATE\_ANDX*.

#### Request Format

```

    UCHAR WordCount;
    UCHAR AndXCommand;
    UCHAR AndXReserved;
    USHORT AndXOffset;
    UCHAR Reserved1;
    USHORT NameLength;
    ULONG Flags;
    ULONG RootDirectoryFid;
    ACCESS_MASK DesiredAccess;
    LARGE_INTEGER AllocationSize;
    ULONG FileAttributes;
    ULONG ShareAccess;
    ULONG CreateDisposition;
    ULONG CreateOptions;
    ULONG ImpersonationLevel;
    UCHAR SecurityFlags;
    USHORT ByteCount;
    ASCII_OR_UNICODE Name[];
  
```

*WordCount* —Must contain the value 24.

*AndXCommand* —Specifies the command code for the next command in the chain, if any.

*AndXReserved* —Is a reserved field and should be zero.

*AndXOffset* —Specifies the offset to the next command in the chain.

*Reserved1* —Is a reserved field, and should be zero. Its purpose is to align the following fields on natural boundaries.

*NameLength* —Specifies the length of the *Name* field, in bytes.

*Note that this structure can't handle very long file names (approaching 4K), because we don't allow the name to be sent as data. The NT Create With SD Or EAs transaction must be used for very long names.*

*Flags* —Specifies the following flags:

*NT\_CREATE\_REQUEST\_OPLOCK* —When set, the client is requesting an oplock.

*NT\_CREATE\_REQUEST\_OPBATCH* —When set, the client is requesting a batch mode oplock.

*NT\_CREATE\_OPEN\_TARGET\_DIR* —Equivalent to *IO\_OPEN\_TARGET\_DIRECTORY* in the NT I/O system.

*RootDirectoryFid* —Specifies the FID for a previously opened directory. If this field is not zero, the file name specified in *Name* is interpreted relative to the specified root directory. If this field is zero, the file name is interpreted relative to the root of the shared resource.

*Note that RootDirectoryFid is subject to change format. While currently 32 bits, it may be changed to a 16 bit field (consistent with Fid fields in other SMBs) and a 16 bit reserved field.*

*DesiredAccess* —Specifies the type of access that the caller requires to the file. The following access types are defined:

*DELETE* —The file may be deleted.

*READ\_CONTROL* —The SD and ownership information associated with the file may be read.

*WRITE\_DAC* —The discretionary SD associated with the file may be written.

*WRITE\_OWNER* —Ownership information associated with the file may be written.

*FILE\_READ\_DATA* —Data may be read from the file.

*FILE\_WRITE\_DATA* —Data may be written to the file.

*FILE\_EXECUTE* —Data may be faulted into memory from the file via paging I/O.

*FILE\_APPEND\_DATA* —Data may only be appended to the file.

*FILE\_READ\_ATTRIBUTES* —File attributes flags may be read.

*FILE\_WRITE\_ATTRIBUTES* —File attributes flags may be written.

*FILE\_READ\_EA* —Extended attributes associated with the file may be read.

*FILE\_WRITE\_EA* —Extended attributes associated with the file may be written.

If the file being created or opened is a directory file, as specified in the *CreateOptions* field, then the following additional types of access may be requested:

*FILE\_LIST\_DIRECTORY* —Files in the directory may be listed.

*FILE\_TRAVERSE* —The directory may be traversed. That is, it may be in the pathname of a file.

*FILE\_READ\_DATA*, *FILE\_WRITE\_DATA*, *FILE\_EXECUTE*, and *FILE\_APPEND\_DATA* accesses are not valid when creating or opening a directory file.

*AllocationSize* —Specifies the initial allocation size of the file, in bytes. This field is ignored unless the file is created, overwritten, or superseded.

*FileAttributes* —Specifies the file attributes for the file. Any combination of flags is acceptable except that all other flags override the normal file attribute, *FILE\_ATTRIBUTE\_NORMAL*. File attributes are only applied to the file if it is created, superseded, or, in some cases, overwritten. See the description in the text below for more details.

The following attribute flags are defined:

*FILE\_ATTRIBUTE\_NORMAL* —A normal file should be created.

*FILE\_ATTRIBUTE\_READONLY* —A read-only file should be created.

*FILE\_ATTRIBUTE\_HIDDEN* —A hidden file should be created.

*FILE\_ATTRIBUTE\_SYSTEM* —A system file should be created.

*FILE\_ATTRIBUTE\_ARCHIVE* —The file should be marked so that it will be archived.

*FILE\_ATTRIBUTE\_CONTROL* —A control file should be created.

*ShareAccess* —Specifies the type of share access that the caller would like to the file. The following share access flags are defined:

*FILE\_SHARE\_READ* —Other open operations may be performed on the file for read access.

*FILE\_SHARE\_WRITE* —Other open operations may be performed on the file for write access.

*FILE\_SHARE\_DELETE* —Other open operations may be performed on the file for delete access.

*CreateDisposition* - Specifies the actions to be taken if the file does or does not already exist. The following disposition values are defined:

*FILE\_SUPERSEDE* —Indicates that if the file already exists then it should be superseded by the specified file. If it does not already exist then it should be created.

*FILE\_CREATE* —Indicates that if the file already exists then the operation should fail. If the file does not already exist then it should be created.

*FILE\_OPEN* —Indicates that if the file already exists it should be opened rather than creating a new file. If the file does not already exist then the operation should fail.

*FILE\_OPEN\_IF* —Indicates that if the file already exists, it should be opened. If the file does not already exist then it should be created.

*FILE\_OVERWRITE* —Indicates that if the file already exists it should be opened and overwritten. If the file does not already exist then the operation should fail.

*FILE\_OVERWRITE\_IF* —Indicates that if the file already exists it should be opened and overwritten. If the file does not already exist then it should be created.

*CreateOptions* —Specifies the options that should be used when creating or opening the file. The following options are defined:

*FILE\_DIRECTORY\_FILE* —On a create operation, indicates that the file being created is a directory file. On an open operation, indicates that the file being opened must be a directory file.

*FILE\_NON\_DIRECTORY\_FILE* —On a create operation, indicates that the file being created is not a directory file. On an open operation, indicates that the file being opened must not be a directory file.

*FILE\_WRITE\_THROUGH* —Indicates that services that write data to the file must actually write the data to the file before the operation is considered to be complete.

*Note that the FILE\_NO\_INTERMEDIATE\_BUFFERING option is not exported via the SMB protocol. The NT redirector should convert this option to FILE\_WRITE\_THROUGH.*

*FILE\_SEQUENTIAL\_ONLY* —Indicates that the file will only be accessed sequentially.

*FILE\_NO\_EA\_KNOWLEDGE* —Indicates that the client does not understand extended attributes.

*FILE\_EIGHT\_DOT\_THREE\_ONLY* —Indicates that the client understands only 8.3 style filenames.

*FILE\_RANDOM\_ACCESS* —Hints that the file is going to be accessed randomly, permitting the server to optimize its behaviour for that case.

*FILE\_DELETE\_ON\_CLOSE* —Indicates that the file is to be deleted when closed.

*ImpersonationLevel* —Specifies the security impersonation level. Must be one of the following values. See the *Distributed System Architecture, Subject Security Context Specification* for a description of their meanings.

*SecurityAnonymous*

*SecurityIdentification*

*SecurityImpersonation*

*SecurityDelegation*

*SecurityFlags* —Specifies the following security flags. See the *Distributed System Architecture, Subject Security Context Specification* for a detailed description of their meanings. The following flag values are defined:

*SMB\_SECURITY\_DYNAMIC\_TRACKING*

*SMB\_SECURITY\_EFFECTIVE\_ONLY*

*ByteCount* —Specifies the length of the byte parameters for the request.

*Name* —Supplies the name of the file to be created or opened. The name is in either ASCII format or Unicode format, depending on the state of flag *SMB\_FLAGS2\_UNICODE* of *Flags2* in the SMB header. The client may use Unicode only if the server indicates in the *Negotiate* response that it supports Unicode. Note that the name string is *not* NUL-terminated.

Response Format

```

UCHAR WordCount;
UCHAR AndXCommand;
UCHAR AndXReserved;
USHORT AndXOffset;
UCHAR OplockLevel;
USHORT Fid;
ULONG CreateAction;
TIME CreationTime;
TIME LastAccessTime;
TIME LastWriteTime;
TIME ChangeTime;
ULONG FileAttributes;
LARGE_INTEGER AllocationSize;
LARGE_INTEGER EndOfFile;
USHORT FileType;
USHORT DeviceState;
BOOLEAN Directory;
USHORT ByteCount;

```

*WordCount* —Must contain the value 34.

*AndXCommand* —Specifies the command code for the next command in the chain, if any.

*AndXReserved* —Is a reserved field and should be zero.

*AndXOffset* —Specifies the offset to the next command in the chain.

*OplockLevel* —Specifies the level of opportunistic lock granted to the opener.

*Fid* —Specifies the identifier for the open file instance.

*CreateAction* —Specifies the action taken with respect to creating or opening the file. This field is valid only if the operation was successful. The following action codes are defined:

*FILE\_SUPERSEDED* —The file existed and was superseded.

*FILE\_CREATED* —The file did not exist and was created.

*FILE\_OPENED* —The file existed and was opened.

*FILE\_OVERWRITTEN* —The file existed and was overwritten.

*FILE\_EXISTS* —The file existed.

*FILE\_DOES\_NOT\_EXIST* —The file does not exist.

*CreationTime* —Specifies the time the file was created.

*LastAccessTime* —Specifies the time the file was last accessed.

*LastWriteTime* —Specifies the time the file was last written.

*ChangeTime* —Specifies the time the file was last changed.

*FileAttributes* —Specifies the file's attributes. For a description of this field, see the description of *FileAttributes* in the request format above.

*AllocationSize* —Specifies the number of bytes allocated to the file.

*EndOfFile* —Specifies the end-of-file offset for the file.

*FileType* — Specifies the type of the file. Possible values are the same as for the like-named field in the SMB *Open And X* command response format:

*FileTypeDisk* —Indicates a disk file or directory as defined in the *FileAttributes* field.

*FileTypeByteModePipe* —Indicates a named pipe.

*FileTypeMessageModePipe* —Indicates a named pipe in message mode.

*FileTypePrinter* —Indicates a printer device.

*FileTypeCommDevice* —Indicates a communications device.

*DeviceState* —If *FileType* is *FileTypeByteModePipe* or *FileTypeMessageModePipe*, this field indicates the state of the IPC device. Otherwise, this field contains garbage. When valid, possible flag values are the same as for the like-named field in the SMB *Open And X* command response format:

*DeviceStateBlocking* —If set, indicates that reads and writes return immediately if no data is available. If clear, reads and writes block if no data is available.

*DeviceStateEndPoint* —If set, indicates the server end of the pipe. If clear, indicates the consumer end of the pipe.

*DeviceStatePipeType* —This is a two-bit field. *00* indicates a byte stream pipe. *01* indicates a message pipe. Other values are undefined.

*DeviceStateReadMode* —This is a two-bit field. *00* indicates that the pipe is to be read as a byte stream. *01* indicates that messages are to be read from the pipe. Other values are undefined.

*DeviceStateIcount* —This is an eight-bit count to control pipe instancing, and is currently not applicable.

*Directory* —Indicates whether the file is a directory.

### Detailed Description

The *NT Create And X* command service either causes a new file (or directory) to be created, or it opens an existing file or device. The action taken is dependent on the name of the object being opened, whether the object already existed, and the specified create disposition value. A file handle is returned that can be used by subsequent service calls to manipulate the file itself or the data within the file.

If *FILE\_APPEND\_DATA* is the only desired-access flag specified, then the caller can only write to the end of the file. Any offset information on writes to the file is ignored. The file will automatically be extended as necessary for these types of write operations.

Specifying the *FILE\_WRITE\_DATA* desired-access flag for a file also allows writes beyond the end of the file to occur. The file is also automatically extended for these types of writes as well.

Access to a file may be shared among separate cooperating processes or threads by requesting that the file system open the file for shared access. This is accomplished through the flags in the *ShareAccess* field. Provided that both file openers have the privilege to access the file in the specified manner, the file can be successfully opened and shared. If the caller does not specify *FILE\_SHARE\_READ*, *FILE\_SHARE\_WRITE*, or *FILE\_SHARE\_DELETE*, then no other open operations may be performed on the file.

In order for the file to be successfully opened, the requested access mode to the file must be compatible with the way in which other opens to the file have been made. That is, the desired access mode to the file must not conflict with the accesses that other openers of the file have disallowed.

The *FILE\_SUPERSEDE* disposition value specifies that if the file does not already exist, it is to be created. If the file already exists, then it should be superseded. Superseding a file requires that the accessor have delete access to the existing file. That is, the existing file is effectively deleted and then recreated. This implies that if someone else already has the file open, they have specified that the file may be deleted by another file opener. This is done by specifying *ShareAccess* with the *FILE\_SHARE\_DELETE* flag set.

The *FILE\_OVERWRITE\_IF* disposition value is much like the *FILE\_SUPERSEDE* disposition value. If the file exists, then it will be overwritten; if it does not already exist then it will be created. Overwriting a file is semantically equivalent to a supersede operation except that it requires write access to the file rather than delete access. That is, the requestor must have write access to the file and if someone else already has the file open, they must have specified that the file may be written by another file opener. This is done by specifying a *ShareAccess* parameter with the *FILE\_SHARE\_WRITE* flag set. Another difference between an overwrite and a supersede is that the specified file attributes are logically OR'd with those already on the file. That is, the caller may not turn off any flags already set in the attributes but may turn others on.

The *FILE\_OVERWRITE* disposition value performs exactly the same operation as a *FILE\_OVERWRITE\_IF*, except that if the file does not already exist the operation will fail.

When opening a named pipe file, the *Name* field of the request should contain only the name of the relative name of the pipe. This contrasts the *Open AndX* and *Open2* SMBs in which the pipe name is prepended by the string "\PIPE\".

### 4.5.3 NT Create With Security Descriptor Or EAs

The *NT Create With SD Or EAs* command is used to create or open a file or a directory, when EAs or an SD must be applied to the file.

*NT Create With SD Or EAs* is implemented as an *NT Transact* function. The function code for *NT Create With SD Or EAs* is *NT\_TRANSACT\_CREATE*.

#### Request Setup Format

None.

#### Request Parameters Format

```

ULONG Flags;
ULONG RootDirectoryFid;
ACCESS_MASK DesiredAccess;
LARGE_INTEGER AllocationSize;
ULONG FileAttributes;
ULONG ShareAccess;
ULONG CreateDisposition;
ULONG CreateOptions;
ULONG SecurityDescriptorLength;
ULONG EaLength;
ULONG NameLength;
ULONG ImpersonationLevel;
UCHAR SecurityFlags;
ASCII_OR_UNICODE Name[];

```

*Flags* —Specifies flags. See the description of the like-named field in *NT Create And X* for the list of valid flags.

*RootDirectoryFid* —Specifies the FID for a previously opened directory. If this field is not zero, the file name specified in *Name* is interpreted relative to the specified root directory. If this field is zero, the file name is interpreted relative to the root of the shared resource.

*Note that RootDirectoryFid is subject to change format. While currently 32 bits, it may be changed to a 16 bit field (consistent with Fid fields in other SMBs) and a 16 bit reserved field.*

*DesiredAccess* —Specifies the type of access that the caller requires to the file. See the description of *NT Create And X* for the list of valid access types.

*AllocationSize* —Specifies the initial allocation size of the file, in bytes. This field is ignored unless the file is created, overwritten, or superseded.

*FileAttributes* —Specifies the file attributes for the file. See the description of *NT Create And X* for the list of valid file attributes.

*ShareAccess* —Specifies the type of share access that the caller would like to the file. See the description of *NT Create And X* for the list of valid share access types.

*CreateDisposition* - Specifies the actions to be taken if the file does or does not already exist. See the description of *NT Create And X* for the list of valid actions.

*CreateOptions* —Specifies the options that should be used when creating or opening the file. See the description of *NT Create And X* for the list of valid options.

*SecurityDescriptorLength* —Specifies the total length in bytes of the security descriptor that should be set on the file, if it is created. The setting of the SD is done as an atomic operation with the creation of the file.

*EaLength* —Specifies the total length in bytes of the list of EAs that should be set on the file if it is created. The setting of the EAs is done as an atomic operation with the creation of the file.

*NameLength* —Specifies the length of the *Name* field, in bytes.

*ImpersonationLevel* —Specifies the security impersonation level. See the description of *NT Create And X* for the list of valid values.

*SecurityFlags* —Specifies flags. See the description of *NT Create And X* for the list of valid flags.

*Name* —Supplies the name of the file to be created or opened. The name is in either ASCII format or Unicode format, depending on the state of flag *SMB\_FLAGS2\_UNICODE* of *Flags2* in the SMB header. The client may use Unicode only if the server indicates in the Negotiate response that it supports Unicode. Note that the name string is *not* NUL-terminated.

### Request Data Format

```
UCHAR SecurityDescriptor[];
UCHAR Padding[];
UCHAR EaList[];
```

*SecurityDescriptor* —The security descriptor to be applied to the file, if it is created. The length of the SD is indicated by the *SecurityDescriptorLength* parameter. If *SecurityDescriptorLength* is zero, the *SecurityDescriptor* and *Padding* fields are not present. The format of *SecurityDescriptor* is the self-relative format described in the *Distributed System Architecture, Object Security Architecture* specification.

*Padding* —Zero to three bytes of padding, to align the *EaList* field on a four-byte boundary. If either of *SecurityDescriptorLength* or *EaLength* is zero, this field is not present.

*EaList* —The list of EAs to be applied to the file, if it is created. The length of the EA list is indicated by the *EaLength* parameter. If *EaLength* is zero, the *Padding* and *EaList* fields are not present. The format of *EaList* is the same as that for *FileFullEaInformation* in the NT I/O system.

### Response Setup Format

None.

## Response Parameters Format

```
UCHAR OplockLevel;  
UCHAR Reserved;  
USHORT Fid;  
ULONG CreateAction;  
ULONG EaErrorOffset;  
TIME CreationTime;  
TIME LastAccessTime;  
TIME LastWriteTime;  
TIME ChangeTime;  
ULONG FileAttributes;  
LARGE_INTEGER AllocationSize;  
LARGE_INTEGER EndOfFile;  
USHORT FileType;  
USHORT DeviceState;  
BOOLEAN Directory;
```

*OplockLevel* —Specifies the level of opportunistic lock granted to the opener.

*Reserved* —Is a reserved field, and should be zero.

*Fid* —Specifies the identifier for the open file instance.

*CreateAction* —Specifies the action taken with respect to creating or opening the file. This field is valid only if the operation was successful. See the description of *NT Create And X* for the list of valid actions.

*EaErrorOffset* —Specifies the offset in bytes into the input EA list of the EA that was invalid. This field is valid only when the input EA list is invalid.

*Need to define which status codes imply that EaErrorOffset is valid.*

*CreationTime* —Specifies the time the file was created.

*LastAccessTime* —Specifies the time the file was last accessed.

*LastWriteTime* —Specifies the time the file was last written.

*ChangeTime* —Specifies the time the file was last changed.

*FileAttributes* —Specifies the file's attributes. For a description of this field, see the description of *FileAttributes* in the NT Create And X request format above.

*AllocationSize* —Specifies the number of bytes allocated to the file.

*EndOfFile* —Specifies the end-of-file offset for the file.

*FileType* — Specifies the type of the file. See the description of *NT Create And X* for the list of valid file types.

*DeviceState* — Indicates the state of an IPC device. See the description of *NT Create And X* for the details of this field.

*Directory* —Indicates whether the file is a directory.

### Response Data Format

None.

### Detailed Description

The *NT Create With SD Or EAs* command is semantically identical to the *NT Create And X* command, with the following additions:

If a security descriptor is supplied, then the SD is applied to the file as an atomic operation. Note that the SD is only set on the file if the file is created, superseded, or overwritten. If setting the SD on the file incurs an error, then the file is not created and an appropriate error is returned.

The security descriptor is a structure of type *SECURITY\_DESCRIPTOR*:

```
typedef struct {
    UCHAR Revision;
    UCHAR Reserved;
    SECURITY_DESCRIPTOR_CONTROL Control;
    PSID Owner;
    PSID Group;
    PACL Sacl;
    PACL Dacl;
} SECURITY_DESCRIPTOR;
```

*Revision* —Contains the revision level of the security descriptor. This allows this structure to be passed between systems or stored on disk even though it is expected to change in the future. The current revision is 1.

*Reserved* —Should be zero.

*Control* —A set of flags which qualify the meaning of the security descriptor or individual fields of the security descriptor.

*SE\_OWNER\_DEFAULTED* —This boolean flag, when set, indicates that the SID pointed to by the *Owner* field was provided by a defaulting mechanism rather than explicitly provided by the original provider of the security descriptor. This may affect the treatment of the SID with respect to inheritance of an owner.

*SE\_GROUP\_DEFAULTED* —This boolean flag, when set, indicates that the SID in the *Group* field was provided by a defaulting mechanism rather than explicitly provided by the original provider of the security descriptor. This may affect the treatment of the SID with respect to inheritance of a primary group.

*SE\_DACL\_PRESENT* — This boolean flag, when set, indicates that the security descriptor contains a discretionary ACL. If this flag is set and the *Dacl* field of the security descriptor is null, then a null ACL is explicitly being specified.

*SE\_DACL\_DEFAULTED* — This boolean flag, when set, indicates that the ACL pointed to by the *Dacl* field was provided by a defaulting mechanism rather than explicitly provided by the original provider of the security descriptor. This may affect the treatment of the ACL with respect to inheritance of an ACL. This flag is ignored if the *SE\_DACL\_PRESENT* flag is not set.

*SE\_SACL\_PRESENT* — This boolean flag, when set, indicates that the security descriptor contains a system ACL pointed to by the *Sacl* field. If this flag is set and the *Sacl* field of the security descriptor is null, then an empty (but present) ACL is being specified.

*SE\_SACL\_DEFAULTED* — This boolean flag, when set, indicates that the ACL pointed to by the *Sacl* field was provided by a defaulting mechanism rather than explicitly provided by the original provider of the security descriptor. This may affect the treatment of the ACL with respect to inheritance of an ACL. This flag is ignored if the *SE\_SACL\_PRESENT* flag is not set.

*SE\_SELF\_RELATIVE* — This boolean flag, when set, indicates that the security descriptor is in self-relative form. In this form, all fields of the security descriptor are contiguous in memory and all pointer fields are expressed as offsets from the beginning of the security descriptor. This flag should always be set in security descriptors within SMBs.

*Owner* — is an offset to a SID representing an object's owner. If this field is null, then no owner SID is present in the security descriptor.

*Group* — is an offset to a SID representing an object's primary group. If this field is null, then no primary group SID is present in the security descriptor.

*Sacl* — is an offset to a system ACL. This field value is only valid if the *SE\_SACL\_PRESENT* control flag is set. If the *SE\_SACL\_PRESENT* flag is set and this field is null, then a null ACL is specified.

*Dacl* — is an offset to a discretionary ACL. This field value is only valid if the *SE\_DACL\_PRESENT* control flag is set. If the *SE\_DACL\_PRESENT* flag is set and this field is null, then a null ACL (unconditionally granting access) is specified.

See *Distributed System Architecture, Security IDentifier Architecture Specification* and *Distributed System Architecture, Access Control List Architecture Specification* for the structure of SIDs and ACLs respectively.

If a list of EAs is supplied, then those EAs are applied to the file as an atomic operation. Note that the EAs are only set on the file if the file is created, superseded, or overwritten. If setting the EAs on the file incurs an error, then the file is not created, an appropriate error is returned, and the response field *EaErrorOffset* is set to the offset into the EA buffer of the EA that caused the error.

The EA list is a sequence of elements of type *FILE\_FULL\_EA\_INFORMATION*:

```
typedef struct {
    ULONG NextEntryOffset;
    UCHAR Flags;
    UCHAR EaNameLength;
    USHORT EaValueLength;
    ASCIIZ EaName[];
    CHAR EaValue[];
} FILE_FULL_EA_INFORMATION;
```

*NextEntryOffset* —Specifies the offset, in bytes, from the start of the current entry to the next entry in the list. If this is the last entry in the list, this field is zero. Each entry in the list must be longword aligned, so *NextEntryOffset* must be a multiple of four.

*Flags* —Specifies flags to be associated with the EA. *FILE\_NEED\_EA* is the only flag currently defined. All other flags should be zero.

*EaNameLength* —Specifies the length of the EA's name field, excluding the NUL termination character.

*EaValueLength* —Specifies the length of the EA's value field.

*EaName* —Specifies the name of the EA in ASCII characters. The name is never in Unicode characters. This field is NUL-terminated.

*EaValue* —Specifies the value of the EA. Note that this field begins immediately after the name field's NUL termination character, without padding for alignment.

#### 4.5.4 NT I/O Control

The *NT I/O Control* command allows device and file system control functions to be transferred transparently from client to server.

This command is implemented as an *NT Transact* function. The function code for *NT I/O Control* is *NT\_TRANSACT\_IOCTL*.

##### Request Setup Format

```
ULONG FunctionCode;  
USHORT Fid;  
BOOLEAN IsFsctl;
```

*FunctionCode* — Specifies the device or file system control code.

*Fid* — Specifies the identifier for the open file instance.

*IsFsctl* — Indicates whether the command is a device control (*FALSE*) or a file system control (*TRUE*).

##### Request Parameters Format

The request parameters buffer is the first buffer.

##### Request Data Format

The request data buffer is the second buffer.

##### Response Setup Format

None.

##### Response Parameters Format

The response parameters buffer is the first buffer.

##### Response Data Format

The response data buffer is the second buffer.

#### 4.5.5 NT Notify Directory Change

The *NT Notify Directory Change* command is used to monitor changes to a directory. The command completes when a change of the specified type is made in the specified directory.

*FILE\_LIST\_DIRECTORY* access to the target directory is required.

The *NT Notify Directory Change* command is implemented as an *NT Transact* function. The function code for *NT Notify Directory Change* is *NT\_TRANSACT\_NOTIFY\_CHANGE*.

#### Request Setup Format

```
ULONG CompletionFilter;
USHORT Fid;
BOOLEAN WatchTree;
UCHAR Reserved;
```

*CompletionFilter* —Specifies a set of flags that indicate the types of operations on the directory or on files in the directory that cause the command to complete. The following flags are defined (all undefined flags should be zero):

*FILE\_NOTIFY\_CHANGE\_FILE\_NAME* —Specifies that the command should complete if a file is added, deleted, or renamed.

*FILE\_NOTIFY\_CHANGE\_DIR\_NAME* —Specifies that the command should complete if a subdirectory is added, deleted, or renamed.

*FILE\_NOTIFY\_CHANGE\_ATTRIBUTES* —Specifies that the command should complete if the attributes of a file or subdirectory are changed.

*FILE\_NOTIFY\_CHANGE\_SIZE* —Specifies that the command should complete if the allocation size or the end of file marker for a file are changed.

*FILE\_NOTIFY\_CHANGE\_LAST\_WRITE* —Specifies that the command should complete if the last write time for a file or subdirectory is changed.

*FILE\_NOTIFY\_CHANGE\_LAST\_ACCESS* —Specifies that the command should complete if the last access time for a file or subdirectory is changed.

*FILE\_NOTIFY\_CHANGE\_CREATION* —Specifies that the command should complete if the creation time for a file or subdirectory is changed.

*FILE\_NOTIFY\_CHANGE\_EA* —Specifies that the command should complete if the EAs for a file or subdirectory are changed.

*FILE\_NOTIFY\_CHANGE\_SECURITY* —Specifies that the command should complete if the security information for a file or subdirectory is changed

*Fid* —Specifies the identifier for the open file instance. The open file must be a directory.

*WatchTree* —Specifies whether all changes to files below the directory should also be reported.

*Reserved* —Is a reserved field and should be zero.

Request Parameters Format

None.

Request Data Format

None.

Response Setup Format

None.

Response Parameters Format

The response parameters buffer is the change data.

Response Data Format

None.

Detailed Description

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

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

When a change that is in the *CompletionFilter* is made to the directory, the command completes. The names of the files that have changed since the last time the command was issued are returned to the client. The *ParameterCount* field of the *NT Transact* response indicates the number of bytes that are being returned. If too many files have changed since the last time the command was issued, then zero bytes are returned and an alternate status code is returned in the *Status* field of the response. If NT status codes were negotiated, then *Status* will contain *STATUS\_NOTIFY\_ENUM\_DIR*. If NT status codes were not negotiated, then *ErrorClass* will be 1 and *Error* will be *ERROR\_NOTIFY\_ENUM\_DIR*, which is a new Win32 error code.

The format of the returned data is defined by the following structure:

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

*NextEntryOffset* —Specifies the offset, in bytes, from the start of the current entry to the next entry in the list. If this is the last entry in the list, this field is zero. Each entry in the list must be longword aligned, so *NextEntryOffset* must be a multiple of four.

*Action* —Specifies what happened to cause this entry to be inserted. The following values are possible:

*FILE\_ACTION\_ADDED* —The file was added to the directory.

*FILE\_ACTION\_REMOVED* —The file was removed from the directory.

*FILE\_ACTION\_MODIFIED* —The file was modified.

*FILE\_ACTION\_RENAMED\_OLD\_NAME* —The file was renamed; this entry supplies the old name.

*FILE\_ACTION\_RENAMED\_NEW\_NAME* —The file was renamed; this entry supplies the new name.

*FileNameLength* —Specifies the length, in bytes, of the name of the file that changed.

*FileName* —Specifies the name of the file that changed. The name is in either ASCII format or Unicode format, depending on the state of flag *SMB\_FLAGS2\_UNICODE* of *Flags2* in the SMB header. The client may use Unicode only if the server indicates in the Negotiate response that it supports Unicode. Note that the name is *not* null-terminated.

If a file is renamed within a single directory, two entries are returned: one specifying the old name of the file and one specifying the new name of the file. If a file is renamed from the directory being monitored to another directory, only a single entry with an *Action* of *FILE\_ACTION\_REMOVED* is returned. If a file is renamed from another directory to the directory being monitored, only a single entry with an *Action* of *FILE\_ACTION\_ADDED* is returned.

#### 4.5.6 NT Cancel

The *NT Cancel* command allows a client to cancel an in-progress request. The request to be canceled is identified by matching the *UID/TID/PID/MID* combination in the header of the original SMB with those in the *Cancel* SMB.

If the specified request is still in progress at the server when the Cancel request is received, the server cancels the request and sends a response to that request with a distinguished status code. The server does not send a response to the cancel SMB.

The command code for *NT Cancel* is *SMB\_COM\_NT\_CANCEL*.

##### Request Format

```
UCHAR WordCount;  
USHORT ByteCount;
```

*WordCount* —Must contain the value 0.

*ByteCount* —Must contain the value 0.

##### Response Format

```
UCHAR WordCount;  
USHORT ByteCount;
```

*WordCount* —Must contain the value 0.

*ByteCount* —Must contain the value 0.

### 4.5.7 NT Query Security Descriptor

The *NT Query Security Descriptor* command allows the client to retrieve the security descriptor on a file.

This command is implemented as an *NT Transact* function. The function code for *NT Query Security Descriptor* is *NT\_TRANSACT\_QUERY\_SECURITY\_DESC*.

#### Request Setup Format

None.

#### Request Parameters Format

```
USHORT Fid;  
USHORT Reserved;  
SECURITY_INFORMATION SecurityInformation;
```

*Fid* —Specifies the identifier of the open file instance.

*Reserved* —Must contain the value 0.

*SecurityInformation* —Flags indicating which security descriptor components are being referenced.

*OWNER\_SECURITY\_INFORMATION* —When set, indicates the owner ID of the object is being referenced.

*GROUP\_SECURITY\_INFORMATION* —When set, indicates the primary group ID of the object is being referenced.

*DACL\_SECURITY\_INFORMATION* —When set, indicates the discretionary ACL of the object is being referenced.

*SACL\_SECURITY\_INFORMATION* —When set, indicates the system ACL of the object is being referenced.

#### Request Data Format

None.

#### Response Setup Format

None.

#### Response Parameters Format

```
ULONG LengthNeeded;
```

*LengthNeeded* —Specifies the size of the data buffer needed to hold the descriptor data.

Response Data Format

The response data buffer is the security descriptor. See the description of *NT Create with Security Descriptor or EAs* for the data format.

#### 4.5.8 NT Set Security Descriptor

The *NT Set Security Descriptor* command allows the client to change the security descriptor on a file.

This command is implemented as an *NT Transact* function. The function code for *NT Set Security Descriptor* is *NT\_TRANSACT\_SET\_SECURITY\_DESC*.

##### Request Setup Format

```
USHORT Fid;  
USHORT Reserved;  
SECURITY_INFORMATION SecurityInformation;
```

*Fid* —Specifies the identifier of the open file instance.

*Reserved* —Must contain the value 0.

*SecurityInformation* —Flags indicating which security descriptor components are being referenced. See the description of the like-named field in *NT Query Security Descriptor*.

##### Request Parameters Format

None.

##### Request Data Format

The request data buffer is the security descriptor. See the description of *NT Create with Security Descriptor or EAs* for the data format.

##### Response Setup Format

None.

##### Response Parameters Format

None.

##### Response Data Format

None.

#### 4.5.9 NT Rename

The NT Rename command allows the client to change the name of a file.

This command is implemented as an *NT Transact* function. The function code for *NT Rename* is *NT\_RENAME*.

##### Request Setup Format

None.

##### Request Parameters Format

```
USHORT Fid;  
USHORT RenameFlags;  
ASCII_OR_UNICODE NewName[];
```

*Fid* —Specifies the identifier of the open file instance.

*RenameFlags* —Specifies flags:

*SMB\_RENAME\_REPLACE\_IF\_EXISTS* —When set, permits overwriting of existing files with the same name as *NewName*.

*Name* —Supplies the name of the file to be created or opened. The name is in either ASCII format or Unicode format, depending on the state of flag *SMB\_FLAGS2\_UNICODE* of *Flags2* in the SMB header. The client may use Unicode only if the server indicates in the Negotiate response that it supports Unicode. Note that the name string is *not* null-terminated. The length of the name is *TotalParameterCount* - *sizeof(Fid)* - *sizeof(RenameFlags)*.

##### Request Data Format

None.

##### Response Setup Format

None.

##### Response Parameters Format

None.

##### Response Data Format

None.

## 5. Outstanding Issues

This section details unresolved issues and questions regarding the NT SMB protocol.

### 5.1 Network Code Page

Do we need to expose client/server/network code page in the protocol, or is it enough to have this available at each node via some other mechanism?

### 5.2 FileLinkInformation

Is adding a link a reasonable candidate for remoting? If so, what restrictions, if any, should be placed on the target name? The I/O specification says that the target name must be fully qualified. What does this mean over the network?

### 5.3 Flags2 vs. NT Create CreateOptions

What happens when the SMB header *Flags2* and *NT Create AndX CreateOptions* fields disagree about long filenames and extended attributes: *SMB\_FLAGS2\_KNOWS\_LONG\_NAMES* vs. *FILE\_EIGHT\_DOT\_THREE\_ONLY* and *SMB\_FLAGS2\_KNOWS\_EAS* vs *FILE\_NO\_EA\_KNOWLEDGE*?

### 5.4 Querying EAs

Do we need a new SMB, or can we do everything with the current SMB? Note that *NtQueryEaFile* is a resumable API, and also allows the caller to start the scan at a specified EA index.

**6. Revision History**

Original (subset) draft, Revision 0.9, March 1, 1991

First widely distributed draft, Revision 1.0, March 15, 1991

Major reduction in scope, Revision 2.0, August 22, 1991

Unknown changes, Revision 2.1, December 10, 1991

Updated to reflect current state of code, Revision 2.2, June 8, 1992

Minor fixes, Revision 2.21, June 11, 1992

Corrections concerning Unicode string alignment and NT Transact responses, Revision 2.22, June 19, 1992

added server CAP\_LOCK\_AND\_READ capability, Revision 2.23, August 10, 1992

corrected SMB File Notify Action values, Revision 2.24, August 11, 1992

added server time fields to Negotiate Response format, Revision 2.25, August 24, 1992