Exhibit B

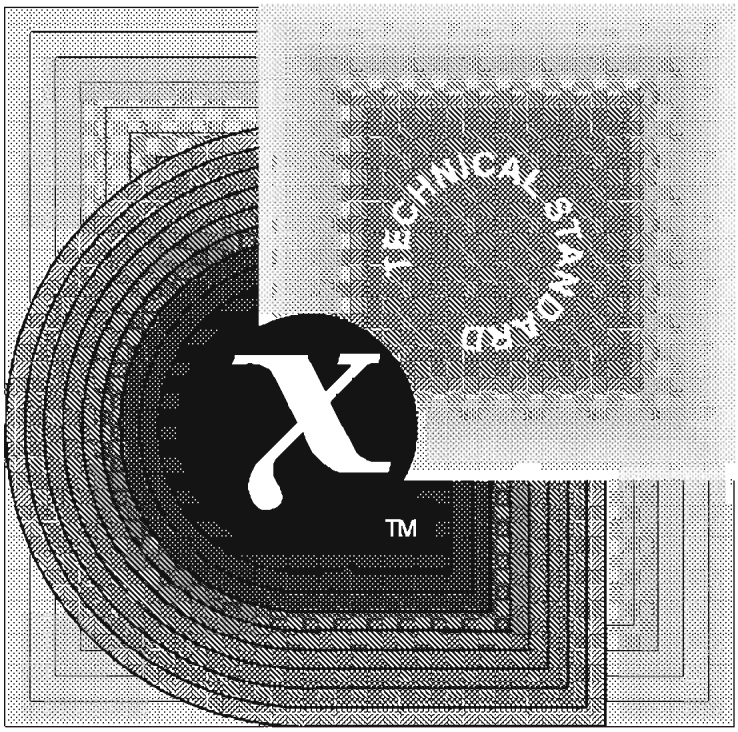
The Open Group, Technical Standard, Protocols for X/Open PC Interworking: SMB, Version 2.0

Customer No.: 8791

Blakely, Sokoloff, Taylor & Zafman, LLP
Sunnyvale, California 94085-4040
Telephone (408) 720-8300
Fax (408) 720-8383

Technical Standard

Protocols for X/Open PC Interworking: SMB, Version 2



THE Open GROUP

[This page intentionally left blank]

X/Open CAE Specification

Protocols for X/Open PC Interworking: SMB, Version 2

X/Open Company Ltd.

X/Open CAE Specification

Protocols for X/Open PC Interworking: SMB, Version 2

ISBN: 1872630456

X/Open Document Number: C209

Published by X/Open Company Ltd., U.K.

Any comments relating to the material contained in this document may be submitted to X/Open at:

X/Open Company Limited
Apex Plaza
Forbury Road
Reading
Berkshire, RG1 1AX
United Kingdom

or by Electronic Mail to:

XoSpecs@xopen.org

Contents

Preface

X/Open

X/Open is an independent, worldwide, open systems organisation supported by most of the world's largest information systems suppliers, user organisations and software companies. Its mission is to bring to users greater value from computing, through the practical implementation of open systems.

X/Open's strategy for achieving this goal is to combine existing and emerging standards into a comprehensive, integrated, high-value and usable open system environment, called the Common Applications Environment (CAE). This environment covers the standards, above the hardware level, that are needed to support open systems. It provides for portability and interoperability of applications, and so protects investment in existing software while enabling additions and enhancements. It also allows users to move between systems with a minimum of retraining.

X/Open defines this CAE in a set of specifications which include an evolving portfolio of application programming interfaces (APIs) which significantly enhance portability of application programs at the source code level, along with definitions of and references to protocols and protocol profiles which significantly enhance the interoperability of applications and systems.

The X/Open CAE is implemented in real products and recognised by a distinctive trade mark — the X/Open brand — that is licensed by X/Open and may be used on products which have demonstrated their conformance.

X/Open Technical Publications

X/Open publishes a wide range of technical literature, the main part of which is focussed on specification development, but which also includes Guides, Snapshots, Technical Studies, Branding/Testing documents, industry surveys, and business titles.

There are two types of X/Open specification:

- *CAE Specifications*

  CAE (Common Applications Environment) specifications are the stable specifications that form the basis for X/Open-branded products. These specifications are intended to be used widely within the industry for product development and procurement purposes.

  Anyone developing products that implement an X/Open CAE specification can enjoy the benefits of a single, widely supported standard. In addition, they can demonstrate compliance with the majority of X/Open CAE specifications once these specifications are referenced in an X/Open component or profile definition and included in the X/Open branding programme.

  CAE specifications are published as soon as they are developed, not published to coincide with the launch of a particular X/Open brand. By making its specifications available in this way, X/Open makes it possible for conformant products to be developed as soon as is practicable, so enhancing the value of the X/Open brand as a procurement aid to users.

- *Preliminary Specifications*

  These specifications, which often address an emerging area of technology and consequently are not yet supported by multiple sources of stable conformant implementations, are released in a controlled manner for the purpose of validation through implementation of products. A Preliminary specification is not a draft specification. In fact, it is as stable as X/Open can make it, and on publication has gone through the same rigorous X/Open development and review procedures as a CAE specification.

  Preliminary specifications are analogous to the *trial-use* standards issued by formal standards organisations, and product development teams are encouraged to develop products on the basis of them. However, because of the nature of the technology that a Preliminary specification is addressing, it may be untried in multiple independent implementations, and may therefore change before being published as a CAE specification. There is always the intent to progress to a corresponding CAE specification, but the ability to do so depends on consensus among X/Open members. In all cases, any resulting CAE specification is made as upwards-compatible as possible. However, complete upwards-compatibility from the Preliminary to the CAE specification cannot be guaranteed.

In addition, X/Open publishes:

- *Guides*

  These provide information that X/Open believes is useful in the evaluation, procurement, development or management of open systems, particularly those that are X/Open-compliant. X/Open Guides are advisory, not normative, and should not be referenced for purposes of specifying or claiming X/Open conformance.

- *Technical Studies*

  X/Open Technical Studies present results of analyses performed by X/Open on subjects of interest in areas relevant to X/Open's Technical Programme. They are intended to communicate the findings to the outside world and, where appropriate, stimulate discussion and actions by other bodies and the industry in general.

- *Snapshots*

  These provide a mechanism for X/Open to disseminate information on its current direction and thinking, in advance of possible development of a Specification, Guide or Technical Study. The intention is to stimulate industry debate and prototyping, and solicit feedback. A Snapshot represents the interim results of an X/Open technical activity. Although at the time of its publication, there may be an intention to progress the activity towards publication of a Specification, Guide or Technical Study, X/Open is a consensus organisation, and makes no commitment regarding future development and further publication. Similarly, a Snapshot does not represent any commitment by X/Open members to develop any specific products.

Versions and Issues of Specifications

As with all *live* documents, CAE Specifications require revision, in this case as the subject technology develops and to align with emerging associated international standards. X/Open makes a distinction between revised specifications which are fully backward compatible and those which are not:

- a new *Version* indicates that this publication includes all the same (unchanged) definitive information from the previous publication of that title, but also includes extensions or additional information. As such, it *replaces* the previous publication.

- a new *Issue* does include changes to the definitive information contained in the previous publication of that title (and may also include extensions or additional information). As such, X/Open maintains *both* the previous and new issue as current publications.

Corrigenda

Most X/Open publications deal with technology at the leading edge of open systems development. Feedback from implementation experience gained from using these publications occasionally uncovers errors or inconsistencies. Significant errors or recommended solutions to reported problems are communicated by means of Corrigenda.

The reader of this document is advised to check periodically if any Corrigenda apply to this publication. This may be done in any one of the following ways:

- anonymous ftp to ftp.xopen.org
- ftpmail (see below)
- reference to the Corrigenda list in the latest X/Open Publications Price List.

To request Corrigenda information using ftpmail, send a message to ftpmail@xopen.org with the following four lines in the body of the message:

```
open
cd pub/Corrigenda
get index
quit
```

This will return the index of publications for which Corrigenda exist. Use the same email address to request a copy of the full corrigendum information following the email instructions.

This Document

Of all the types of computers, personal computers are the most abundant. Originally intended to be a personal productivity tool, an ever-increasing number of them are being connected to computer networks, thus becoming parts of distributed information systems.

Personal computers normally run under single-user operating systems with interfaces differing from those specified in the X/Open Portability Guide. However, X/Open realises how important it is to facilitate interworking between personal computers and X/Open-compliant systems in a standardised way.

Two areas have to be addressed to achieve this goal; interoperability, and programming interfaces to server functions facilitating applications portability. Interoperability means that personal computers and X/Open-compliant systems can interchange information using the same network protocols. Standardisation of programming interfaces to server functions, in addition to standardisation of protocols, makes it possible to write distributed client/server applications whose server component will be portable to all X/Open-compliant systems.

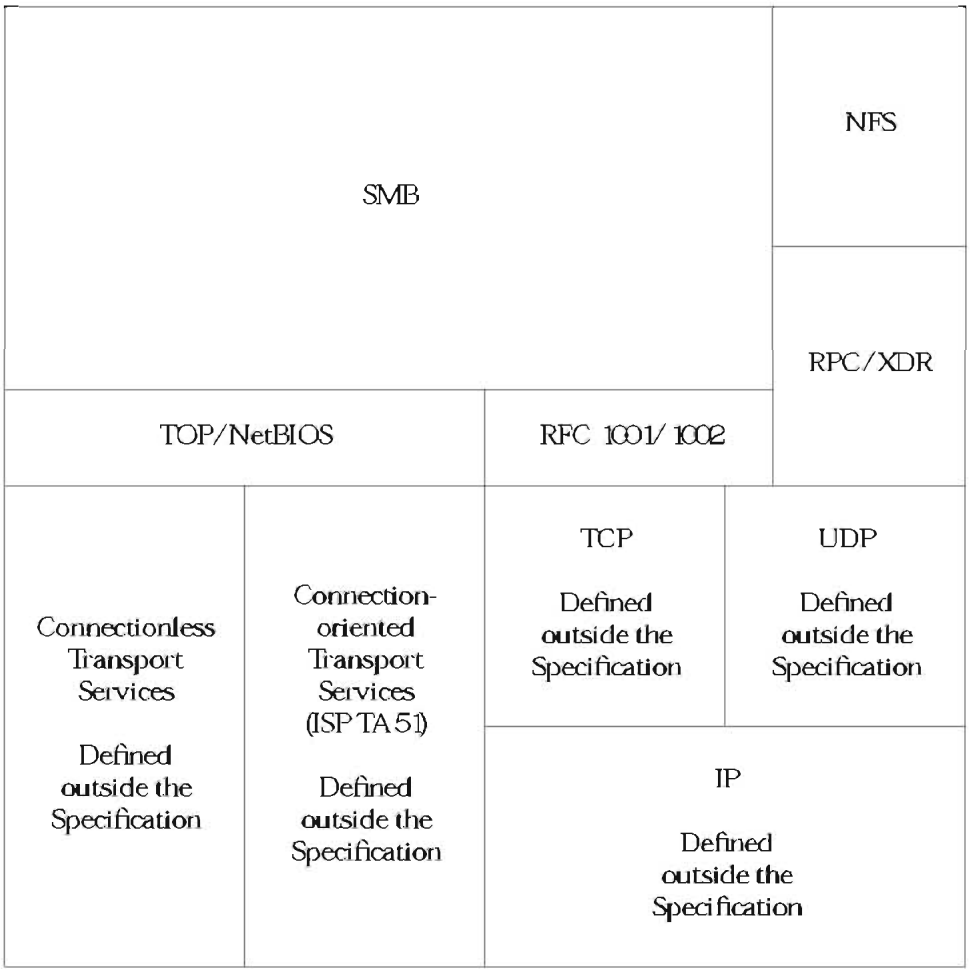
For interoperability via asynchronous serial links, X/Open has already defined in the X/Open Portability Guide, Issue 3 a file transfer protocol and a set of features provided on X/Open-compliant systems for terminal emulators. Now it is time to address interworking in local area networks (LANs).

In the X/Open (PC)NFS and SMB Developers' Specifications interoperability of personal computers and X/Open-compliant systems is addressed. The applications portability components, containing definitions of programmatic interfaces to server functions, are documented in the X/Open CAE Specification, IPC Mechanisms for SMB and the X/Open CAE Specification, Use of XTI to Access NetBIOS.

When connecting personal computers and X/Open-compliant systems via standard transport protocols, there appear to be two possibly overlapping but distinct market segments. In the first one, personal computers are added to existing networks of X/Open-compliant systems which already have a distributed file system, the most widely-adopted one being the Network File System originally designed by Sun Microsystems. In the second one, X/Open-compliant servers are added to LANs consisting primarily of personal computers. For personal computers running under DOS or OS/2 operating systems, which is the vast majority, the generally accepted non-proprietary protocol is the Server Message Block from Microsoft Corporation.

Therefore, for connecting personal computers to X/Open-compliant systems, both the (PC)NFS (see the X/Open Developers' Specification, Protocols for X/Open PC Interworking: (PC)NFS) and the SMB protocols have been adopted by X/Open.

The following diagram illustrates the relationship of the service protocols (defined in the X/Open (PC)NFS and SMB Developers' Specifications) to their underlying transport protocols. It also reflects the organisation of the two documents. The (PC)NFS specification describes the protocols for NFS, RPC and XDR. The SMB specification describes the protocols for SMB, the mapping of NetBIOS over an OSI transport (TOP/NetBIOS) and the mapping of NetBIOS over an Internet Protocol Suite transport (RFC 1001/RFC 1002).



Since SMB and NFS protocols do not easily map onto the seven layer OSI Reference Model, the diagram does not use it.

Throughout the specification "DOS" is used to refer to the MS-DOS or PCDOS personal computer operating system.

Trade Marks

Ethernet® is a registered trade mark of Xerox Corporation.

LAN Manager™ is a trade mark of Microsoft Corporation.

MS-DOS® is a registered trade mark of Microsoft Corporation.

NFS® is a registered trade mark of Sun Microsystems.

OS/2® is a registered trade mark of International Business Machines Corporation.

Palatino® is a registered trade mark of Linotype AG and/or its subsidiaries.

PC-NFS™ is a trade mark of Sun Microsystems.

UNIX® is a registered trade mark of UNIX System Laboratories Inc. in the U.S.A. and other countries.

X/Open™ and the "X" device are trade marks of X/Open Company Ltd. in the U.K. and other countries.

Referenced Documents

The following documents are referenced in this specification:

JPC
: X/Open CAE Specification, JPC Mechanisms for SMB
(Document No.: C 195, ISBN: 1-872630-28-6).

NetBIOS
: X/Open CAE Specification, Use of XTI to Access NetBIOS, contained in X/Open CAE Specification, X/Open Transport Interface (XTI)
(Document No.: C 196, ISBN: 1-872630-29-4).

OS/2
: Microsoft OS/2 Programmer's Reference, Volume 4.

(PC)NFS
: X/Open Developers' Specification, Protocols for X/Open PC Interworking: (PC)NFS
(Document No.: D030, ISBN: 1-872630-00-6).

SMB
: X/Open Developers' Specification, Protocols for X/Open PC Interworking: SMB
(Document No.: D 110, ISBN: 1-872630-01-4).

XNFS
: X/Open CAE Specification, Protocols for X/Open Interworking: XNFS, Issue 4
(Document No.: C218, ISBN: 1-872630-66-9).

XPG3
: X/Open Portability Guide, Issue 3, January 1989

Chapter 1

Introduction

1.1 Why Republish

A previous version of this specification has been published. The previous version described the SMB protocol up to a dialect level called extended. Since that time, a new dialect has been added and several errors and omissions were found in the specification. This version of the specification corrects the errors and omissions and contains the definition for the extended 2.0 SMB dialect. The extended protocol of the previous version of this document is now called extended 1.0 which is to be distinguished from the new extended 2.0 dialect.

1.2 This Document

The relevant parts of this CAE Specification include the specification of the SMB protocol itself, definition of the conventions used in mapping SMB redirector semantics onto X/Open semantics, specifications of the binding of the NetBIOS interface to popular protocol stacks, and selection of protocol profiles to permit interoperability.

Information regarding NetBIOS is provided because the great majority of SMB redirector implementations of the SMB protocols rely on NetBIOS as well.

The interface to the NetBIOS implementation on the CAE system is outside the scope of this specification. Within this document only the NetBIOS service definition to the Internet Protocol Suite (RFC 1001/1002) (see Appendices F and G) and an OSI transport (TOP/NetBIOS) (see Appendix E on page 281) are considered.

In this second publication, the SMB definitions necessary for Inter-process Communication (IPC) from SMB redirectors to processes executing on the same CAE system as the LMX server have been removed. These definitions are found in the X/Open CAE Specification, IPC Mechanisms for SMB.

This specification does include the SMB protocol and the SMB service definition to be implemented by an LMX server. The SMB service definition of the SMB redirector as well as user interfaces necessary to access network resources are outside the scope of this specification.

1.3 Overview of Document Layout

Chapter 2 provides an overview of the service and security model for the SMB protocol.

Chapter 3 discusses the conventions related to the rules the SMB protocol maintains. This chapter describes the environments maintained within the SMB protocol model as well as rules governing file locking and user security.

Chapter 4 describes conventions that can be followed for mapping the SMB protocol model described in Chapter 3 into the CAE environment. This chapter provides guidelines for such things as how filenames in the CAE environment are viewed by the SMB protocol environment.

Chapter 5 defines the basic structure, data items and constant definitions for the SMB protocol.

The core dialect is defined in Chapter 6 through Chapter 9.

Additions to the core dialect that make up the core plus dialect are found in Chapter 10.

Chapter 11 through Chapter 14 define the extended 1.0 SMB dialect.

The additions for the extended 2.0 SMB dialect are covered in Chapter 15 and Chapter 16.

A description of the mapping of DOS and OS/2 system calls to SMB protocol requests, descriptions of support of NetBIOS names on TCP/IP and OSI protocols, and additional SMB protocols that may be used for LMX server administration are contained in the appendices to this specification.

Chapter 2

SMB File-sharing Service Model

This CAE Specification describes the X/Open LAN Manager (LMX) architecture, the Server Message Block (SMB) protocol, and their applicability to interoperability between X/Open-compliant LAN Manager implementations running in an X/Open Common Applications Environment (CAE) and SMB redirectors running DOS or OS/2.

LMX provides a file and print-sharing service which preserves, as far as possible, the same semantics as provided by a DOS or OS/2 system to an application. This service is provided by mapping the SMB redirector semantics onto those supported by the CAE system in which the LMX server runs.

This model is in contrast to a file-sharing service, in which the LMX server provides a complete emulation of the SMB redirector's file storage architecture, but does not permit access to that emulation from applications running on the same CAE system. The intent behind the LMX approach is to permit applications existing on SMB redirectors and CAE systems to cooperate in the processing of information. Within this architecture the SMB redirector can assume that only the file contents are stored in the same format as in the SMB redirector's operating system. That is, directory information does not need to be stored on the CAE system in a file or have the same layout as in the SMB redirector's operating system.

In LMX resources are *shared* by making the name of the resource available for access from the network. For example, the LMX server named XOPEN will make a resource DOCUMENTS that contains this document available. This allows users on SMB redirectors to connect to this resource and access this data. In this example the resource DOCUMENTS could point to a directory tree that contains the files belonging to this document. The user will see this directory and its files as if they are on the local SMB redirector's system.

2.1 SMB Protocol Principles

File and print sharing are implemented using the SMB protocol. This protocol is used between two types of system: SMB redirectors and LMX servers. When a user on an SMB redirector wants to make use of SMB file and print services available in the network the user needs an SMB redirector implementation of the SMB protocol. Upon request the SMB redirector will connect to an LMX server. Throughout this document the term LMX server does not imply any particular design.

The SMB protocol requires a reliable connection-oriented virtual circuit provided by a NetBIOS implementation.

Each LMX server in the network will offer resources. When a user on an SMB redirector wishes to use a resource, or resources, from an LMX server, the user of the SMB redirector will cause the SMB redirector to set up a single LMX session with the desired LMX server using NetBIOS. The action of setting up the LMX session includes using NetBIOS to locate the system in the network then negotiating the level of SMB support desired by the SMB redirector. If multiple resources are desired by the SMB redirector, the SMB redirector will use the single LMX session to perform all SMB exchanges. So, if the user requests use of both a file system share and a printer share on the same LMX server, then only one LMX session exists between the SMB redirector and this LMX server system.

Once the LMX session has been established the SMB redirector will take initiative to request services offered by the LMX server by sending SMB requests across the LMX session. Each SMB request is executed by the LMX server and the result is sent back to the SMB redirector in an SMB response. SMB redirector implementations may support multiple simultaneous connections to different LMX servers.

The SMB protocols can be divided into:

- core protocol
- core plus protocol
- extended 1.0 protocol
- extended 2.0 protocol

each one being a superset of the previous one. The extended protocols offer a richer set of functionality and are required for some of the IPC mechanisms described in the X/Open CAE Specification, IPC Mechanisms for SMB.

In the extended protocols, mechanisms exist to have users authorised by the LMX server (see Section 2.2). If an SMB protocol supporting user authorisation is negotiated the LMX server will authorise the one user working on the SMB redirector upon request of the SMB redirector. This is commonly referred to as a *logon procedure*.

Once the level of protocol is negotiated, and if necessary the user has been authorised, the SMB redirector will request access to a specific resource. The resource requested may be a directory tree, spooled device, I/O device, etc. If the requested resource has been made available by the LMX server for access by that user, file and spool operations can be executed (for example, open file, show print queue) from now on.

2.2 Security Overview

The networks using the SMB protocol will contain not only multi-user systems with user-based security models, but also single-user systems that have no concept of user IDs or permissions. Once these systems are connected to the network, however, they are in a multi-user environment and need a method of access control. First, unsecure systems need to be able to provide some sort of *bona fides* to other systems in the network which do have permissions. Second, unsecure nodes need to control access to their resources by others.

The SMB protocol defines a mechanism that enables the network software to provide the security where it is missing from the operating system, and supports user-based security where it is provided by the operating system. The mechanism also allows systems with no concept of user ID to demonstrate access authorisation to systems which do have a permission mechanism.

The LMX server will define the security mode that is being used; it cannot be negotiated by the SMB redirector. Within the SMB protocols two forms of security exist:

- share-level security mode

  Can be applied to restrict the access to a *shared* resource, placing access control at the level of the resource.

- user-level security mode

  Can assign user context to anyone establishing an LMX session. This way different access rights can be granted to people connecting to the same resource. This form of security can only be used when an extended SMB protocol has been negotiated.

2.2.1 Share-level Security Mode

A share-level security mode LMX server makes a resource available to all users on the network. Any user who knows the name of the LMX server, the name of the resource, and the password, has the same access to everything (for example, read-only) within a resource. The password is optional.

For example, the LMX server named XOPEN offers the resource DOCUMENTS. This is a file system subtree where each individual file or directory will have the same permissions for all users, for example, read-only or read/write. Access to this resource is controlled by a password. The LMX server could make a second resource available with a different password and different access rights pointing to the same directory with the files belonging to this document.

2.2.2 User-level Security Mode

A user-level security mode LMX server also makes a resource available, but in addition requires the user to provide a username and optional password in order to gain access.

Thus the LMX server is now able to allow differing access rights depending on the validated user. The access rights may not only be specified per resource but may be set individually for each file or directory accessible via a resource name. One user may have full access, another read-only and perhaps another no access to different files and directories within the shared resource.

For example, on the LMX server named XOPEN with the resource DOCUMENTS a user called BOB could be the author of the document and a user called JAN a reviewer for the document. Now BOB can have read/write access to the document while JAN is only able to read the files belonging to the document.

Chapter 3

SMB Protocol Conventions

Much of the SMB protocol definition is design and implementation-independent. In addition to the SMB protocol and specific meaning of fields, the LMX server has to obey certain rules. This chapter includes a summary of SMBs and defines generic conventions for LMX servers, such as:

1. SMB Environments
2. user-level and share-level security modes
3. connection protocols
4. naming
5. wildcards and the interpretation of wildcard pathnames
6. file attributes
7. locking, including opportunistic locking, and an implicit variety of locking for enhancing the performance of applications which do not make explicit lock requests
8. chaining, and the mechanism for making multiple requests in a single SMB
9. exception and error handling
10. timeouts
11. downward-compatibility support

3.1 Summary of SMBs

The following table lists the SMBs (requests and responses) which are required for various levels of the SMB protocol. The table gives the name of each request/response and a brief description, the section of this specification in which the SMB is described, and indicates whether the request is part of the core (C), core plus (C+), extended 1.0 (E) or extended 2.0 (E2) SMB protocol. The SMBs used to implement file and print sharing are defined here. Additional SMBs can be found in the X/Open CAE Specification, IPC Mechanisms for SMB and the appendices to this specification.

In the following tables, the SMB names ending with capital X indicate that the SMB request permits chaining (see Section 3.9 on page 22).

Name	Description	Section	Protocol
SMBchkpath	Verify path is directory	8.7	C
SMBclose	Close file	7.10	C
SMBcopy	Copy file	14.1	E
SMBcreate	Create/Open file	7.1	C
SMBdskattr	Get the LMX server file system information	8.6	C
SMBecho	Test an LMX session	14.2	E
SMBexit	Indicate process exit	6.4	C
SMBfclose	Close active search	13.2	E
SMBffirst	Active search	13.1	E
SMBfindclose	Close an active search	15.4*	E2
SMBfindnclose	Notification of close for an active search	15.3*	E2
SMBflush	Flush data for file(s)	7.9	C
SMBfunique	One-time active search	13.3	E
SMBgetatr	Get file attributes	8.4	C
SMBgetattrE	Get extended file attributes	13.4	E
SMBlock	Lock byte-range of file	7.7	C
SMBlockingX	Lock multiple ranges and X	12.2	E
SMBlockread	Lock and read byte-range	10.3	C+
SMBlseek	Set current file pointer	7.6	C
SMBmkdir	Create new directory	8.1	C
SMBmknew	Create new file	7.2	C
SMBmove	Move files by copying	14.4	E
SMBmv	Change name of file(s)	7.11	C
SMBnegprot	Negotiate Protocol	6.1	*
SMBopen	Open File	7.3	C
SMBopenX	Extended open and X	12.1	E
SMBread	Read from file	7.4	C
SMBreadbmpx	Read block multiplexed	12.5	E
SMBsecpkgX	Negotiate security packages and X	11.2	E
SMBtrans2(TRANSACT2_FINDFIRST)	Active search	16.3	E2
SMBtrans2(TRANSACT2_FINDNEXT)	Active search	16.4	E2
SMBtrans2(TRANSACT2_MKDIR)	Create new directory	16.13	E2
SMBtrans2(TRANSACT2_OPEN)	Open File	16.2	E2
SMBtrans2(TRANSACT2_SETFSINFO)	Set file system information	16.6	E2
SMBtrans2(TRANSACT2_QPATHINFO)	Query file information	16.7	E2
SMBtrans2(TRANSACT2_SETPATHINFO)	Set file information	16.8	E2
SMBtrans2(TRANSACT2_QFILEINFO)	Query file information	16.9	E2
SMBtrans2(TRANSACT2_SETFILEINFO)	Set file information	16.10	E2
SMBtrans2(TRANSACT2_FINDNOTIFYFIRST)	Monitor file or directory changes	16.11	E2
SMBtrans2(TRANSACT2_FINDNOTIFYNEXT)	Continue monitoring	16.12	E2

(*) The *SMBnegprot* response changes if either extended dialect of SMB is being negotiated.

Name	Description	Section	Protocol
SMBreadbraw	Read block raw	10.1	C+
SMBreadX	Read and X	12.3	E
SMBrmdir	Delete empty directory	8.2	C
SMBsearch	Directory wildcard lookup	8.3	C
SMBsesssetupX	Session setup and X	11.3	E
SMBulogoffX	User logoff and X	15.5[4]	E.2
SMBsetatr	Set file attributes	8.5	C
SMBsetattrE	Set extended file attributes	13.5	E
SMBsplclose	Close and queue spool file	9.3	C
SMBsplopen	Create spool file	9.1	C
SMBsplretq	Get spool queue info	9.4	C
SMBsplwr	Write to spool file	9.2	C
SMBtcon	Tree connect	6.2	C
SMBtconX	Tree connect and X	11.4	E
SMBtdis	Tree disconnect	6.3	C
SMBunlink	Delete file	7.12	C
SMBunlock	Unlock byte-range of file	7.8	C
SMBwrite	Write to file	7.5	C
SMBwritebmpx	Write block multiplexed	12.6	E
SMBwritebraw	Write block raw	10.2	C+
SMBwriteclose	Write and close file	10.5	E
SMBwriteunlock	Write and unlock byte-range	10.4	C+
SMBwriteX	Write and X	12.4	E

3.2 SMB Environment Definitions

The following environments are defined for the purpose of specifying the SMB protocol. An LMX server does not need to construct such an environment, as long as the required semantics are preserved.

The hierarchy of environments is summarised below:

LMX Session Environment
  User Environment (UID)
  Resource Environment (TID)
    Process Environment (PID)
    Multiplex Request Environment (MID)
    File Environment (FID)

1. LMX Session Environment

   This consists of one LMX session established between an SMB redirector and an LMX server. The LMX session represents the logical connection between the SMB redirector and the LMX server. This connection is initiated by the SMB redirector and is only considered an LMX session after the *SMBnegprot* protocol exchange has successfully completed. Only one protocol dialect can be negotiated on a single LMX session.

   An LMX session is implemented using a NetBIOS session.

   For each LMX session the maximum buffer size for subsequent SMB requests and responses is set by the LMX server and sent to the SMB redirector. It is the SMB redirector's responsibility not to send larger SMB requests than expected by the LMX server.

   An LMX server may drop the LMX session after the last resource environment has been terminated. When an LMX session becomes inactive for some period of time and the LMX server is not maintaining any file environment information for the SMB redirector, the LMX server may choose to terminate the LMX session. This allows other SMB redirectors to connect and use the LMX session resource. It is the responsibility of the SMB redirector to reestablish the LMX session after it has been terminated due to this timeout.

   If the LMX session environment is terminated, all PIDs, TIDs and FIDs within it will be invalidated.

2. User Environment, also called the Logon Environment

   This is represented by a user ID (UID). A UID uniquely identifies a user within a given LMX session environment. Within dialects of this document, there is exactly one UID per LMX session. An LMX server executing in user-level security mode uses this to identify the scope and type of access allowed for this user. In share-level security mode this environment is not used.

   If the user environment is terminated in the extended 2.0 dialect via *SMBulogoffX*, all FIDs and TIDs currently held by the UID are invalidated. In the extended 1.0 dialect no termination SMB exists other than the termination of the LMX session.

3. Resource Environment

   This is represented by a TID. A TID uniquely identifies a resource being shared within the LMX session between the SMB redirector and the LMX server. The TID is requested by the SMB redirector and assigned by the LMX server. The resource being shared may be a directory tree, spooled device, I/O device, etc. More than one TID may exist within a single LMX session environment.

In an LMX server executing in share-level security mode, the TID also identifies the scope and type of accesses allowed across the connection.

Within the core SMB protocol it is possible for the LMX server to set a new maximum buffer size for subsequent SMB requests within this resource environment. The new maximum buffer size is not only valid for the new resource environment, but for all resources environments established within the LMX session. It is the SMB redirector's and the LMX server's responsibility not to send larger SMBs than negotiated.

If a resource environment is terminated (via an *SMBtdis* request) all PIDs and FIDs within it will be invalidated. The LMX server will close all files, free all locks, release all active file searches and terminate all processes created on behalf of that TID.

4 Process Environment

This is represented by a process ID (PID). A PID uniquely identifies an SMB redirector process or thread within a given LMX session environment. Most SMB requests include a PID to indicate which process initiated the request. SMB redirectors inform LMX servers of the creation of a new process by simply introducing a new PID. The LMX server does not maintain any process relationships.

Within the core SMB protocol the *SMBexit* request terminates the process environment. Otherwise, there is no mechanism for the LMX server to determine a process exit on the SMB redirector. It is the SMB redirector's responsibility to close a resource when the last SMB redirector process referencing the resource closes it.

Files opened by one process may be manipulated by another process in the same resource environment (that is, possessing the same TID).

If in the SMB core protocol a process environment is terminated, the LMX server will invalidate all FIDs created by that PID.

5 File Environment

This is represented by a file ID (FID). An FID identifies an open file and is unique within a given LMX session environment. Another LMX session environment may be given an FID of the same value, but the FID will refer to a different open instance of the same or different file. The scope of the FID is the user environment. This means a file may be opened and its FID passed to another process (using a different PID in the same LMX session) for use without being opened by this process. The second process must use the same UID and TID as the process which opened the file.

If a file environment is terminated (via an SMB request) or invalidated, all locks placed on that FID will be released.

6 Multiplexed Request

This is represented by a multiplexed ID (MID). This is not an environment, but a part of the SMB request that needs to be discussed at this time. An MID uniquely identifies an SMB request within the LMX session. By using the MID, an SMB redirector is able to send multiple requests to the LMX server and determine which SMB response is associated with each SMB request. There is no termination of the Multiplex Request Environment. It is maintained for the SMB redirector's use only. The core and core plus protocol do not use an MID.

3.3 Share-level and User-level Security Modes

3.3.1 Share-level Security Mode

The following section applies to the access of LMX servers that use share-level security. By default all SMB requests are refused as unauthorised. When an administrator of the LMX server chooses to allow access to resources, he or she would establish each *share* with the following attributes:

- The resource type (see Section 5.3.6 on page 45) that will be used in *SMBtcon* and *SMBtconX* requests.
- The mapping of the resource type to the resource on the CAE system (for example, file system subtrees will be identified on the CAE system with the root of the offered subtree being the directory shared).
- An indication of which access to this resource is permitted (for example, read-only).
- Optionally, a password (to be supplied in the *SMBtcon* or *SMBtconX* request) is required before access to the resource is permitted.

Note that when a file system subtree is shared, all files underneath that directory are then affected. If a particular file is within the range of multiple offers, connecting to any of the offers gains access to the file; the access rights gained (for example, read *versus* read/write) will depend upon the attributes of the offer that the SMB redirector connected to. The LMX server will not check for nested directories with more restrictive permissions.

For example, if the LMX server is offering a read/write share JAZZ, corresponding to path `/usr/jazz`, and a read-only share JAZZCAT, corresponding to path `/usr/jazz/catalog`, an SMB redirector which connected to the JAZZ share would be permitted read/write access to the file `catalog/myrecs`, even though that file is also contained within the scope of a read-only share.

3.3.2 User-level Security Mode with Extended Protocols

LMX servers with user-based file security (in user-level security mode) will require the SMB redirector to present a username and password (if any) along with the requested UID value prior to accessing resources.

A username and password are sent by the SMB redirector and validated by the LMX server via the *SMBsesssetupX* protocol. If the username and password are valid the LMX server responds with a UID that is used to identify the user on all subsequent SMB requests and prove to the LMX server that this user has been authenticated. The SMB redirector must associate the UID with the user and include the UID for all network resource accesses made by that user.

The *SMBtcon* and *SMBtconX* protocols are still used to define the directory subtree or other resource available to the user, but the LMX server uses the UID to allow differing types of access to the same resources under a given TID. Note that a single SMB redirector may issue multiple *SMBtcon* or *SMBtconX* in order to gain access to multiple shared resources.

An LMX server in user-level security mode will still require administrative action to make a share available. The attributes of the share are the same as for share-level security mode, except that a single password is no longer used for the share.

If the LMX server responds to an *SMBnegprot* request and selects the extended protocol, it will indicate in the SMB response the security mode in effect. This allows the SMB redirector to know whether the User Logon information is needed in the *SMBsesssetupX* request.

Each LMX server may maintain a list of valid users. It may then verify every access by these users.

From the LMX server's point of view, the UID is therefore not associated with a particular shared resource, but with the authenticated user. The UID may be used to access any shared resource controlled by the LMX server which has been connected to via the TREE CONNECT[1] protocol.

3.3.3 User-level Security with Core Protocol

There is no support within the core protocol to allow user-level security for SMB redirectors that are only capable of working with the core protocol. An LMX server in user-level security mode may decline connections with an SMB redirector requesting only the core protocol.

In an effort to be flexible, the LMX server may select to support the core-only SMB redirector by mapping the SMB redirector into the user-level security environment. This mapping could be performed by the following steps:

1. If the SMB redirector's system name is defined as a username (and the password supplied with *SMBtcon* matches), the user logon will be performed using that value.
2. If the above fails, the LMX server may reject the request or assign a default username (probably allowing limited access).
3. The UID will then be ignored and all access will be validated assuming the username selected above.

The above allows LMX servers in user-level security mode'' to accommodate SMB redirectors supporting only the SMB core protocol.

1 The term TREE CONNECT is used to represent either the *SMBtcon* or *SMBtconX* request usage.

3.4 Connection Protocols

No network traffic is generated when an LMX server makes resources available for sharing. The required information is simply stored until requests from SMB redirectors arrive.

The SMB protocol makes use of a NetBIOS transport facility. NetBIOS defines a set of network transport facilities. The interface is outside the scope of this document. The NetBIOS functions can be implemented over a variety of transport protocols, however within this document only the mapping of NetBIOS over TCP and UDP (see Appendices F and G) and NetBIOS over ISO transport services (see Appendix E on page 281) are considered.

To establish an LMX session the SMB redirector will establish a NetBIOS session with the LMX server. Therefore the LMX server listens on the LMX NetBIOS name (see Section 3.5 on page 15).

After the LMX session has been established the SMB redirector will negotiate the SMB protocol level sending an *SMBnegprot*. The *SMBnegprot* must be the first SMB request sent on the NetBIOS session. In the *SMBnegprot* response the LMX server will specify the maximum buffer size that the SMB redirector is allowed to request or send. Due to the nature of the NetBIOS transport service the maximum buffer size will be in the range of 1K to 64K bytes. Each SMB request or response will be sent as a single NetBIOS message.

When the user of the SMB redirector issues a command to connect to a particular share, the SMB redirector generates an *SMBtcon* or *SMBtconX* request containing the name of the shared resource and the associated password. The password could be empty. If the LMX server is in user-level security mode the username and password will be supplied via the *SMBsesssetupX* request. If no *SMBsesssetupX* request is received, the LMX server may use the SMB redirector's system name as described in Section 3.3.3 on page 13 to perform user authorisation.

When running in share-level security mode, on receiving the *SMBtcon* or *SMBtconX* request, the LMX server verifies the resource name/password combination and returns either an error code or an identifier (the TID).

The resource name is included in the TREE CONNECT request and the identifier (TID) identifying the connection is returned. The meaning of this identifier (TID) is LMX server-specific; the SMB redirector must not associate any specific meaning to it.

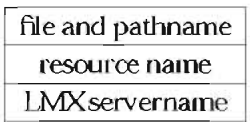
The SMB redirector must associate the identifier with the device name being redirected (specified by the user in the command which initiated the TREE CONNECT) and include the TID for all future network resource accesses.

3.5 Naming

Within the SMB protocols three types of name formats can be distinguished:

- NetBIOS names
- names according to the Uniform Naming Convention (UNC)
- long filenames

An LMX server supports the following hierarchy of names for file and print sharing:



The first layer, the *LMX servername*, is used by the SMB redirector to identify the specific LMX server desired. This LMX servername is typically used by the user on the SMB redirector when he wants to connect to a particular resource maintained by that LMX server. The mapping of the LMX servername to the NetBIOS name may be obtained by converting the LMX servername to upper case, padding up to the fifteenth byte with 0x20 and adding 0x20 in the sixteenth byte. This approach restricts the length of the LMX servername to 15 characters.

3.5.1 Resource Names

Each LMX server supports a collection of *resource names*. A resource name represents a resource provided by the LMX server. This name is at a minimum in 8.3 format (refer to Section 3.5.3 on page 16), however, actual restrictions on this name are implementation-specific. Examples of resources are:

- file system subtrees
- printers
- IPC facilities (outside the scope of this specification, see the X/Open CAE Specification, IPC Mechanisms for SMB)
- administrative data, which can be accessed and modified via remote administration (see Appendix B on page 263)
- directly accessible devices (outside the scope of this specification)

A resource name is also commonly referred to as a *share name*. The resource name for IPC facilities *IPC$* and the resource name for administrative data *ADMIN$* are reserved and cannot be used for other services.

3.5.2 NetBIOS Names

NetBIOS names are used to establish a NetBIOS session between the LMX server and the SMB redirector, the LMX session. Other NetBIOS names are used for messaging services, as described in the X/Open CAE Specification, IPC Mechanisms for SMB. A NetBIOS name has a length of 16 bytes. NetBIOS names have no structure; that is, there is no concept of network number, host number, socket number, and so on. Each participant in a communication uses a NetBIOS name. NetBIOS names are dynamically claimed and relinquished. There are two types of NetBIOS name: unique, which can be claimed by only one system at a time, and group, which can be claimed by several systems at a time.

Since NetBIOS names are used to connect systems with the SMB protocol, some structure on the NetBIOS name is imposed. For the LMX servername, the first fifteen bytes normally comprise

the LMX servername in all upper-case characters. Any remaining bytes are padded with trailing blanks (ASCII 0x20) to bring the total length of the NetBIOS name to 15 bytes. LMX servernames are usually simple, unstructured names, such as XOPEN-PC1G, TOOLSVR, JASONZ.

The sixteenth byte is used to distinguish various uses of the SMB protocol, as follows:

0x00 Used by the SMB redirector to name its end of a file-sharing connection; also used for the sending end of messaging circuits and the sending and receiving ends of class 2 mailslot datagrams (see the X/Open CAE Specification, IPC Mechanisms for SMB). A NetBIOS name ending in 0x00 is also said to be in redirector format.

0x20 Used by LMX servers as the NetBIOS name to which they listen for incoming connections (LMX network name). A NetBIOS name ending in 0x20 is also said to be in server format.

It is important to note that a single system may use all forms at various times, depending upon the type of interaction and the system with which it is interacting.

So, as an example, the SMB redirector will use a NetBIOS name ending in 0x00 as the caller name and a NetBIOS name ending in 0x20 for the LMX servername.

3.5.3 Uniform Naming Convention

UNC names are constructed from names having an 8.3 format that are separated by a backslash (\). An 8.3 format name consists of two components: a one to eight-byte basename must be present and an optional one to three-byte extension may be added. If the second component is specified, the two components are separated by a period (.), hence the term 8.3 format. Within an 8.3 format name the following bytes are illegal:

- " . / \ [ ] : | < > + = ; , * ? 0x20 (space)
- bytes less than 0x20

Note that the characters * and ? are used in some SMB requests as wildcard characters.

3.5.4 Canonical Pathnames

For all of the dialects defined in this document, except for the extended 2.0 SMB protocol, file and directory names need to follow the Uniform Naming Convention (UNC). The backslash (\) separator is the directory separator. Two special directory names, . and .., must be recognised. They have the usual CAE meanings; . points to its own directory, .. points to its parent directory. In the root directory of the file system subtree, . and .. are not present.

Note that it is the LMX server's responsibility to ensure that virtual root as defined by the TID.

3.5.5 Long Names

The extended 2.0 protocol allows for the creation of long file and directory names with a total length up to 255 characters. These names are case-insensitive and may be case-preserving (implementation-dependent). That is, the names File and file will represent the same name. Long names have a free format, compared to UNC names. It is possible to create a long name for a file which contains multiple instances of the component separator .. Directories are still delimited by the \ character.

3.6 Wildcards

Some SMB requests support wildcard filenames as the last 8.3 or long filename format of a pathname. These are filenames which refer to a number of files based on a pattern-match defined by the wildcard string. Only filenames which are acceptable under the filename convention (see Section 4.2 on page 28) can be matched by wildcards.

Each part of an 8.3 format name - the basename and the extension (if applicable) - is treated separately. For long filenames the . in the name is significant even though there is no longer a restriction on the size of each of the components on either side of the ..

- The * character matches an entire part, as will an empty specification of that part. If received, it is interpreted to mean filling the remainder of the component in the name with ? and performing the search with this wildcard character. Any characters that occur after the * are ignored.
- The ? character matches exactly one character. Multiple ? characters at the end of a part match that number of characters or fewer.

For example, the strings ABC.TXT and A.TXT would match the wildcard *.TXT, but ABC.T would not; AB.C and ABC.C would match A??.C, but ABCD.C would not; *.* would match all filenames.

Some SMBs, such as *SMBmv* and *SMBcopy*, use wildcards to transform filenames. In this case, two wildcard patterns would be supplied; the non-special characters in filenames matching the first wildcard would be replaced with the non-special characters in the same relative positions from the second wildcard, and the wild fields would be left unchanged.

For example, the wildcards *.F and *.FOR would transform ABC.F to ABC.FOR, but ABC.F1 would not match the first wildcard and would not be transformed; A?B??.C and X?Y??.TXT would transform A1B2C to X1Y2TXT, but A1B234.C would not match the first wildcard.

3.7 File Paradigm

All resource type information is stored using a file paradigm. For the resource type the following file types are defined:

- regular files on file system subtrees
- spool files for printers

Other types defined that are outside the scope of this specification are:

- named pipes for IPC facilities
- mailslots for IPC facilities
- devices on directly accessible devices

Note that directories are never treated as files, but require special SMB requests to be read.

3.7.1 Regular Files

In SMB requests the following attributes are known:

read-only file	If this attribute is set, write access is denied. Otherwise read and write access is allowed.
hidden file	The file is excluded from normal directory searches.
system file	The file is excluded from normal directory searches.
volume ID	11-byte volume label to identify a file system subtree. It is implemented as a special file and must reside on the root directory of the file system subtree. Some SMB redirectors expect this to be a file.
directory	The file is a directory.
archive file	If this attribute is set it indicates that the file has been changed since the last backup. Typically it is set whenever the file has been written to and will be cleared by backup programmes.

The volume ID attribute cannot be specified together with other attributes. The other attributes can be set concurrently. Files without any attribute set are referred to as regular files.

3.7.2 Open Modes

There are two groups of file exclusion which can be selected via the SMB protocol when a file is opened. A file opened in any deny mode may be opened again only for accesses allowed by the deny mode. The two groups and their subtypes are:

Group 1

DENY NONE Anyone else may read and/or write.

DENY ALL Deny other users any access to this file.

DENY READ Other users may access for writing.

DENY WRITE Other users may access for reading.

The deny modes provide exclusion at the file level. A file opened in any deny mode may be opened again only for the access allowed by the deny mode. This exclusion applies to all subsequent opens of the file even if it is from the same process requesting the original deny mode open. The DENY READ and DENY ALL modes deny opening a file for execution (reference Section 5.3.5 on page 44).

Subsequent opens of a file may specify more restrictive deny modes as long as the new exclusions do not conflict with the existing deny modes granted.

The following table outlines access to the file:

Existing Deny Mode	access	New open requesting DENY ALL	DENY WRITE	DENY READ	DENY NONE
DENY ALL	R/W	fail	fail	fail	fail
	READ	fail	fail	fail	fail
	WRITE	fail	fail	fail	fail
DENY WRITE	R/W	fail	fail	fail	READ
	READ	fail	READ	fail	READ
	WRITE	fail	fail	READ	READ
DENY READ	R/W	fail	fail	fail	WRITE
	READ	fail	WRITE	fail	WRITE
	WRITE	fail	fail	WRITE	WRITE
DENY NONE	R/W	fail	fail	fail	ALL
	READ	fail	ALL	fail	ALL
	WRITE	fail	fail	ALL	ALL

Group 2

Compatibility — Within an LMX session, once a file has been opened in compatibility mode, all subsequent opens of that file by any process must be in compatibility mode until the last open instance has been closed. If a process opened a file for any access, another process using the same LMX session may open the same file for any access.

Across LMX sessions, compatibility mode opens are mapped as follows:

Compatibility Read Only	<>	DENY WRITE
Compatibility Write Access	<>	DENY ALL

The rules for group 1 open modes apply.

3.7.3 Write Behaviour

The SMB protocols make assumptions on the state of written data; that is, whatever data is written is assumed to be what will be read at a later instant. The actual placing of the data onto the storage medium is a function of the LMX server. Yet, the SMB protocols do allow the SMB redirector to make suggestions about the placing of the data.

There are two types of write behaviour:

Write through — The data is to be placed on the storage medium prior to the response to the write request.

Write behind — It is acceptable to cache the data internally to the server and respond to the write request immediately.

These write behaviour modes are only availabe in the extended dialects of the SMB protocols. The core and core plus dialects assume a write through behaviour.

3.8 Locking Conventions

3.8.1 Byte Locking

The SMB protocol supports a form of record locking for read access or write access. This lock covers a range of bytes and cannot overlap any other locked range. Access to a locked range of bytes from a process which did not obtain the lock is prevented. Processes need not take a lock to determine if any other process had that range locked as well.

3.8.2 Opportunistic Locking

Opportunistic locking is a performance enhancement available in the extended protocols which enables an SMB redirector to reduce the number of SMB requests to a minimum when it is the only SMB redirector accessing a file opened in non-exclusive mode. This form of locking allows the SMB redirector to cache locking requests as long as no other process is attempting to access the file. The support of opportunistic locking is the one instance within the SMB protocols where the LMX server will make requests of the SMB redirector.

An SMB redirector requests an opportunistic lock (or oplock) in two ways:

1. by setting bit 5 (and optionally bit 6 for additional notifications such as file deletion) in the *smb_flg* field of the SMB header (see Section 5.1 on page 37) of the *SMBopen*, *SMBcreate* or *SMBmknew* core SMB requests. The oplock is granted by bit 5 being set in the *smb_flg* field of the SMB response. If bit 5 is not set in the response then the oplock was not granted.
2. by setting bit 1 (and optionally bit 2) of the *smb_flags* field in the *SMBopenX* extended SMB request. The oplock is granted by bit 15 of *smb_action* being set in the response.

An opportunistic lock may only be granted if no other SMB redirector has the file open. An LMX server need not implement opportunistic locking; such an implementation would simply deny all oplock requests.

The LMX server must break the oplock and notify the SMB redirector in the following cases:

- another process attempts to open the file
- if bit 6 and bit 2 were set in the oplock request and an operation that changes the file (for example, *SMBunlink*, *SMBmv*, *SMBmove*) was received by the LMX server

When an LMX server decides to break an oplock, it must perform the following steps:

1. Hold off the request which caused it to break the oplock.
2. Send to the SMB redirector which has the oplock an *SMBlockingX* request with MID = – 1.
3. Permit the SMB redirector to flush any data that was cached by sending the appropriate SMB WRITE requests. The SMB redirector must flush any cached byte-range locks as well. These lock requests can be embedded in the *SMBlockingX* request which must be issued in response to the broken oplock notification.
4. Finally, the SMB redirector sends an *SMBlockingX* request responding to the request issued in step 1. If the *SMBlockingX* request contained any lock requests, a response by the LMX server must be generated. If the request did not contain lock requests, no response by the LMX server is generated. Note that the *SMBlockingX* request should contain no unlock requests, as the SMB redirector was not explicitly locking to the LMX server while it had an opportunistic lock.

The SMB redirector with the oplock may choose to close the file during step 3 processing. If it does so, the LMX server may grant an opportunistic lock to the new requesting SMB redirector if all other conditions are met.

If the SMB redirector has issued an SMB CLOSE request on the file at the same time the LMX server has attempted to break the oplock, the SMB redirector will ignore the *SMBlockingX* request; the LMX server must handle the SMB CLOSE request correctly and not expect a response to its attempt to cancel the oplock.

It is possible that notification of a broken oplock (the *SMBlockingX* request), and some other request from the SMB redirector, cross on the network. In this case, the LMX server must note that the notification is outstanding and cause all SMB requests to fail (by returning zero-length data, for example). The SMB redirector will respond to the broken oplock notification and retry the SMB request.

An LMX server is permitted to detect access to an opportunistically-locked file from an LMX server-resident process and break the lock; however, this functionality is not mandatory.

3.9 Chaining of Extended SMB Requests

Certain extended SMB protocol requests (those whose names end with X) can have an additional SMB request chained to them; however, each SMB request which permits chaining allows only a subset of the possible SMB requests to be chained. The chaining of SMB requests allows for a reduction in the number of request/response actions that need to be taken in some instances. For example, if an application on the SMB redirector requests a lock of a byte range followed by a read of the data in this byte range, the SMB redirector may choose to cache the sending of the locking request until the actual read occurs then send an *SMBlockingX*, *SMBreadX* chained request.

The following rules must be obeyed by chained SMB requests:

1. The chained SMB request does not repeat the SMB header information. Rather, it starts with its own *smb_wct* field. The *smb_com2* field in each *SMB...X* request specifies the SMB command code for the chained SMB request.
2. All chained SMB requests and their data must fit within the negotiated maximum buffer size. This size limitation also applies to the amount of data in the SMB request.
3. There is one SMB request sent containing the chained SMB requests and there is one SMB response to the chained SMB requests. The LMX server must not elect to send separate SMB responses to each of the chained SMB requests.
4. All chained SMB responses must fit within the negotiated maximum buffer size. This limits the maximum value on an embedded READ, for example. It is the SMB redirector's responsibility not to request more bytes than will fit within the multiple SMB response.
5. If the last request of a chained series is a chained SMB request (that is, *SMB...X*), the *smb_com2* field must be 0xOOff (also referred to as the NIL command).
6. The LMX server will implicitly use the result of the prior SMB requests in chained SMB requests. For example, the TID obtained via *SMBtconX* would be used in a chained *SMBopenX*, and the FID obtained in the *SMBopenX* would be used in a chained *SMBread*. If chained requests reference an FID, the *smb_fid* field in each SMB request must contain the same FID value. In other words, each SMB request can only reference the same FID (and TID) as the other SMB request in the combined request. The chained SMB requests can be thought of as performing a single (multi-part) operation on the same resource.
7. The first SMB request to encounter an error will stop all further processing of chained SMB requests. The LMX server shall not undo SMB requests that succeeded.

   Suppose *SMBopenX* and *SMBread* were requested; if the LMX server were able to open the file successfully but the read encountered an error, the file would remain open. This is exactly the same as if the SMB requests had been sent separately.
8. If an error occurs while processing chained SMB requests, the SMB response element of the chained SMB responses in the buffer will be the one which encountered the error. Other unprocessed chained SMB requests will have been ignored when the LMX server encountered the error and will not be represented in the chained SMB response. More specifically, the last valid *smb_com2* (if not the NIL command) will represent the SMB command code on which the error occurred. If no valid *smb_com2* is present, then the error occurred on the first SMB request and *smb_com* contains the SMB command code which failed. In all cases, the error class and code are returned in the *smb_rcls* and *smb_err* fields of the SMB header at the start of the SMB response.
9. Each chained SMB request and SMB response contains the offset (from the start of the SMB header) to the next chained SMB request/response in its own *smb_off2* field. This permits

chained SMB requests to be built without packing them. There may be space between the end of the previous SMB request (as defined by *smb_wct* and *smb_bcc*) and the start of the next chained SMB request; this simplifies the building of chained SMB requests.

10. The data in each SMB response is expected to be truncated to the negotiated maximum number of 512 byte blocks which will fit (aligned at a 32-bit boundary) in the maximum buffer size, with any remaining bytes in the final buffer.

3.10 Exception and Error Handling

Exception handling within the SMB environment is built upon the various environments (see Section 3.2 on page 10). When any environment is terminated in either an orderly or disorderly fashion, all contained environments are terminated.

3.10.1 Disorderly LMX Session Dissolution

The rules for disorderly LMX session termination are as follows:

- An LMX server may terminate the LMX session to an SMB redirector at any time if the SMB redirector is generating invalid SMB requests. However, wherever possible the LMX server should first return an error code to the SMB redirector indicating the cause of the LMX session abort.
- If an LMX server gets a hard error on an LMX session (such as a send failure) all LMX sessions from that SMB redirector may be aborted.

An SMB redirector is expected to reestablish an LMX session in the case where it was dropped by the LMX server due to inactivity.

On write-behind activity, a subsequent WRITE or CLOSE of the file will return the fact that a previous WRITE failed. Normally, write-behind failures are limited to hard disk errors and file system out-of-space conditions.

3.10.2 Errors and Error Handling

In the case of success for file and print sharing, the LMX server must return error class SUCCESS and error code SUCCESS. For situations where no error is defined by the SMB protocol, the error class ERRSRV and error code ERRerror are to be returned.

The contents of SMB response parameters other than the SMB header fields are not guaranteed in the case of an error return. In particular, the LMX server may choose to return only the SMB header portion from the SMB request in the SMB response; that is, the SMB header fields *smb_wct* and *smb_bcc* (see Section 5.1 on page 37) may both be zero (0).

3.11 Timeouts

The extended protocols provide for timeouts on the LMX server. SMB requests which may timeout include:

- opens to directly accessible devices
- byte-range locking
- read or write on directly accessible devices, mailslots and named pipes (refer to the X/Open CAE Specification, IPC Mechanisms for SMB)

If an LMX server cannot support timeouts, then the error <ERRSRV, ERRtimeout> is returned, just as if a timeout had occurred, if the resource is not available immediately upon request. A timeout can indicate a delay time, an indefinite delay, or that a system default should be used. Default timeouts apply to direct access devices, mailslots and named pipes only.

3.12 Downward-compatibility Support

The core and extended SMB protocol requests and responses are variable length. Thus additional fields may be added in the *smb_vwv[]* and the *smb_buf[]* areas in future dialects (see Section 5.1 on page 37). LMX servers must be implemented such that additional fields in either of these areas will not cause the SMB request to fail. If additional fields are encountered, which are not recognised by the LMX server's level of implementation, they should be ignored. This allows for future upgrade of the SMB protocol and eliminates the need for reserved fields.

Chapter 4

LMX Considerations

This chapter highlights possible behaviours of LMX servers and deals with aspects that are caused by hosting LMX servers in the CAE.

The conventions an LMX server must adhere to are:

1. user mapping from SMB redirectors to CAE environment
2. filename mapping, which defines the mapping from the namespace provided by the SMB canonical pathname format to the namespace of CAE
3. access and attribute mapping, which defines the mapping from CAE access rights to SMB file attributes and *vice versa*
4. locking, which defines the mapping from the SMB-supported locking operations to those locking operations supported by CAE

Other items where LMX servers may choose differing approaches are:

1. SMB protocol dialect (or dialects) and password encryption
2. consequences of the CAE file system
3. LMX server caching
4. method of support for printer spooling
5. usage of the underlying network, including the choice of the network protocol, interoperability with other file-sharing principles and extensions beyond a single subnetwork

4.1 LMX Username Mapping

CAE file system security is based on a user or process having a CAE UID and one or more CAE GIDs (refer to the X/Open Portability Guide, Issue 3, Volume 2, XSI System Interface and Headers). Personal computers remotely accessing a CAE file system via an LMX server must not compromise the CAE file system security.

An LMX server must provide a mechanism to map a user to a CAE UID and CAE GIDs. This mapping may be different for share-level and user-level security mode (refer to Section 3.3 on page 12). For example, an LMX server running in user-level security mode may map each user to its own unique CAE UID and CAE GIDs, while an LMX server running in share-level security mode may map all users to a common CAE UID and CAE GIDs. This mapping of a username and password into the CAE environment may use the CAE user account system to hold the usernames and passwords. Or, there may be a separate user account system for users of SMB redirectors that maps these users into the CAE environment. Regardless of the approach taken, an LMX server must guarantee that a user does not have any more access permissions than a CAE process with the same CAE UID and CAE GIDs.

When running in user-level security mode, the UID used in the SMB requests may be relative to the LMX session. The LMX server therefore needs to map each pair (LMX session, UID) to the individual CAE UID and CAE GIDs.

4.2 LMX Filename Mapping

This convention governs the mapping between SMB pathnames (see Section 3.5.4 on page 16) and names maintained in the file system on the CAE system. The *SMBsesssetupX* request uses a bit (bit 4 in the *smb_flg*; see Section 5.1 on page 37) in the SMB header which indicates whether or not the pathnames in subsequent SMB requests have been translated to SMB canonical pathnames. LMX servers must support this bit being set.

In addition to this flag, in the extended protocols another bit (bit 3 in the *smb_flg*) in the SMB header indicates whether the SMB redirector desires case-insensitive pathnames. If this bit is set, operations should be case-insensitive. LMX servers must support this bit being set.

If an LMX server does not support the functionality of either bit 3 or bit 4 when not set, the server may choose to ignore these bits and attempt to use the pathname provided in the SMB request in the manner it would for the condition where the bits are set. This means that when an SMB redirector performs a request with one (or both) of these bits cleared and the server does not support that form of pathname, the SMB redirector will receive an error condition produced by the normal functioning of the LMX server (that is, file not found).

With regard to both these flags, the LMX server must generate pathnames in SMB responses which match the requested form. If the SMB redirector did not request canonical pathnames, the LMX server must not map pathnames in responses, but simply use the local representation.

Pathnames following the Uniform Naming Convention (see Section 3.5.4 on page 16) from the SMB redirector side are to be mapped by the LMX server into the CAE file system. Characters with values larger or equal to 0x80 may not be supported or converted from upper to lower-case (and *vice versa*) by LMX servers. All other characters are mapped according to the following rules:

1. Filenames with . and extension are used as is.
2. Convert all characters of value less than 0x80 to lower case (unless case-sensitive mode was requested).
3. The directory separator \ is converted to /.
4. Accept the special names . and .. as is.
5. Leave any other special characters as they are. If any forbidden characters (see below) remain in a name, reject the request.

Names of files on the CAE system are mapped by the LMX server to canonical pathnames according to the following rules. An LMX server implementation may map a wider range of CAE filenames into a canonical pathname bypassing some of the restrictions below. However, all mappings need to obey rules one to three.

1. Names which are all lower case are split into filename and extension at the first period (.). If case-insensitive mode was requested, all characters of value less than 0x80 are converted to upper case.
2. The special files . and .. are not translated and are used as is.
3. The directory separator / is converted to \.
4. If case-insensitive mode was requested, names containing an upper-case letter are invisible and inaccessible from the SMB redirector. If case-sensitive mode was requested files of mixed case are visible to the SMB redirector.
5. Basenames longer than 8 characters are invisible and inaccessible from the SMB redirector depending on the dialect chosen. The extended 2.0 dialect allows for longer file and

directory names.

6. Names containing a leading . (that is, a null basename part) are invisible and inaccessible from the SMB redirector.
7. Names containing a trailing . (that is, a null extension with an extension separator present) are invisible and inaccessible from the SMB redirector.
8. Names containing more than one . are invisible and inaccessible from the SMB redirector.
9. Names containing more than three characters following a . are invisible and inaccessible from the SMB redirector.
10. Names containing characters not permitted in canonical pathnames are invisible and inaccessible from the SMB redirector. Those illegal characters are:

    "." (as anything but a separator for the extension)
    " " (the space character, ASCII 0x20)
    any value less than ASCII 0x20
    0x2B "+", 0x5B "[", 0x5D "]", 0x2A "*", 0x3F "?", 0x3A ":", 0x5C "\",
    0x3B ";", 0x2F "/", 0x3D "=", 0x3C "<", 0x3E ">", 0x22 """, 0x7C "|",
    0x2C ","

Examples:

CAE filename	SMB redirector (case-insensitive mode)
a	A
acn	ACN
main.c	MAIN.C
123456789	<not accessible: too long>
12345678	12345678
/users/acn/main.c	\USERS\ACN\MAIN.C
file.	<not accessible: trailing dot>
MSnet	<not accessible: upper-case letter>
ACN	<not accessible: upper-case letter>
file.baad	<not accessible: extension too long>
s.c.x	<not accessible: too many dots>

4.3 LMX File Mapping

4.3.1 SMB File Attributes

SMB file attributes (see Section 3.7 on page 17) are not the same as CAE file attributes. The mapping of the read-only and directory attributes is the minimum set of required functionality. Any other attributes not supported by the LMX server may be ignored. If the read-only attribute is specified, the SMB redirector has no write permission. For files created, the LMX server will turn off the CAE write permission. If the directory attribute is specified, the requested name will map to a CAE directory. LMX servers may support more SMB file attributes but are not allowed to use different semantics for the read-only and directory attribute.

Changing the read-only atribute via *SMBsetatr* or *SMBsetattrE* will affect the write mode of the file from the LMX server's perspective; hence, in user-level security mode the UID specified must map to that of a CAE process with appropriate privilege.

4.3.2 CAE File Access Permissions

CAE provides a *umask* (refer to the X/Open Portability Guide, Issue 3, Volume 2, XSI System Interface and Headers) to define the default file access permissions to be used when a new file is created. An LMX server must provide a mechanism to define the *umask* to be used for CAE files created on behalf of the users. The mechanism is implementation-dependent. For example, an implementation may provide a common *umask* for all users or may define a *umask* per user.

In CAE environments, it is necessary to have both the read and search attributes on a directory to be allowed to view and transverse the directory (refer to the X/Open Portability Guide, Issue 3, Volume 2, XSI System Interface and Headers). An LMX server must provide support that allows for SMB redirectors to create directories that can be viewed and transversed.

When the LMX server opens a file on behalf a user (that is,the SMB redirector's user mapped to a CAE UID and CAE GIDs) the CAE access permissions for that file must be obeyed.

4.3.3 File System Issues

CAE provides a method whereby the maximum allowed size of an individual file can be controlled. This control is provided via *ulimit* (refer to the X/Open Portability Guide, Issue 3, Volume 2, XSI System Interface and Headers). An LMX server may provide support where this feature can be used to govern the maximum file size allowed for all users of the LMX server or even individual users.

If this support is provided, it is not possible to retrieve the value for *ulimit* from SMB redirectors. Therefore, SMB redirectors cannot tell the difference between a file size restriction or a file system being out-of-space. The manner by which an LMX server handles the CAE *ulimit* feature is implementation-dependent.

The LMX server will report either the free space of a single file system or the total free space of all file systems that the shared file system subtree, accessible from the SMB redirector, may span. Thus it is possible to get into a state where a directory path on the LMX server has run out of free space, but another directory path has not. In this state, SMB redirectors will report to the user that there is free space available on the server and yet the user will not be able to write data to files on the file system subtree or *vice versa*.

It is possible in a CAE environment that the LMX server has no control over the creation time given to a particular file. Therefore, support for the setting of the creation time provided by an SMB redirector is implementation-dependent.

When returning available space on the LMX server to the SMB redirector (see Section 8.6 on page 107), it may be necessary for the SMB server to report an allocation unit that is larger than the 512-byte units of the CAE system in order to avoid overflowing the number of allocation units available in the SMB response. This can result in a rounding error for the free space information.

Some CAE systems provide no way for a program to block until the local file cache has actually flushed to the disk, but simply indicate that a flush has been scheduled and will complete soon. An LMX server should nonetheless take steps to maximise the probability that the data is truly on disk before the SMB redirector is notified.

4.3.4 CAE Special Files

LMX servers may allow access to CAE special files, such as CAE-defined FIFOs or character and block special files (refer to the X/Open Portability Guide, Issue 3, Volume 2, XSI System Interface and Headers). Support for special file access is not a requirement for LMX servers.

4.3.5 Deleting or Renaming a File

The specification for deleting or renaming a file via an SMB request (for an example, see Section 7.12 on page 92 or Section 7.11 on page 89) specify that for a file to be deleted no other process may have the file open. In a CAE environment, it may not be possible for the LMX server to determine whether another CAE application has the file to be deleted open. Therefore, it is implementation-dependent whether the LMX server will not allow an SMB redirector to delete or rename a file while another CAE application has the file open for use. Additionally, it is possible for a CAE application to delete or rename a file while an SMB redirector has the file open for use. The actions taken by the LMX server under these circumstances are implementation-dependent.

4.3.6 Long Filenames

When using the extended 2.0 protocol dialect, an LMX server may support the use of long filenames. These are filenames which do not conform to the 8.3 format (refer to Section 3.5.5 on page 16). It is possible that the CAE system on which the LMX server is executing does not support filenames to the maximum length allowed in the long filename definition. In this case, the LMX server may support names longer than the 8.3 format yet restrict the maximum length of the name to the length supported by the CAE system. As an example, suppose the CAE system supports names up to fourteen characters in length. An LMX server on this system is allowed to provide long name support to the SMB redirectors and restrict the maximum length of such names to fourteen characters. It is not required that an LMX server supporting long filenames guarantees support of the maximum name length in the long filename definition.

4.3.7 Extended Attributes

The extended 2.0 protocol allows for the storage and retrieval of extended attributes on a file stored on the LMX server. Extended attributes are *name=value* pairs where the length of the combination of the *name=value* pair will not exceed 65535 bytes. Both the *name* and the *value* portion of the pair are free format and application-specific. The application will store and retrieve the information based on the *name*. Support for extended attributes is optional.

Some SMB redirectors will store a collection of default extended attributes (EAs) when the support for extended attributes is provided by the LMX server. Known examples of names and values for EAs stored are:

.COMMENTS= An ASCIIZ string giving some general discussion on the contents of the file.

.HISTORY=	An ASCIIZ string indicating creation and change history for the file.
.KEYPHRASES=	A collection of key words or phrases that pertain to the file.
.SUBJECT=	A subject line for the file.
.TYPE=	The type of the file; that is, it is a document file, plain text or a spreadsheet.

For moving or copying files in an environment where LMX servers may or may not be supporting EAs, SMB redirectors will copy all of the data contents of a file between servers and warn the user about loss of EA information. The specifics of the SMB error codes that must be supported by the LMX server to generate this warning are discussed in Chapter 16 on page 207.

4.4 LMX File Locking

The locking model and functionality provided by the SMB protocols (and thus expected by SMB redirector processes) and the model being used by applications running in a CAE environment are quite different. This mismatch makes it impossible to require an LMX server to properly mediate interlocking between an SMB redirector process and CAE application accessing the same file.

Some forms of interlocking mediation are possible. If an LMX server chooses to support file locking, it should support at least the features described in this section.

The SMB protocol does deny modes on open (see Section 3.7.2 on page 18) and byte-range locks. The core SMB protocol supports only one type of byte-range lock via the *SMBlock* request that excludes that byte-range from any other lock, read or write access by other SMB redirectors. The extended protocols support additionally read-only locks via *SMBlockingX*.

The CAE does not define any forms of deny mode as in the SMB protocols. The CAE, however, specifies two forms of locks (see the X/Open Portability Guide, Issue 3, Volume 2, XSI System Interface and Headers):

F_RDLCK Lock byte range allowing multiple readers (shared lock); a process may write to the range (with or without an *F_RDLCK*) if no other process has an *F_RDLCK* on that range. The file must have been opened with read access.

F_WRLCK Lock byte range allowing R/W (read and write) for locking process only (exclusive lock). The file must have been opened with write access.

These locks are advisory, rather than mandatory. With advisory locking, cooperating processes must acquire locks to determine whether any other process has locked that range as well.

4.4.1 Interlocking Behaviour

Deny Modes

An LMX server must mediate deny modes between multiple SMB redirector processes. But it cannot completely enforce those access denials against other LMX server-resident applications, since those other processes may not be making lock requests against the file, and the CAE does not provide a mandatory locking function. LMX servers may provide some forms of deny-mode between an SMB redirector and a CAE application.

When interlocking for deny modes is supported, the LMX server may place the following locks when an SMB redirector requests a byte-range lock:

SMB requested mode	Action
Opens for DENY ALL with all access modes, DENY WRITE with READ access mode, and COMPATIBILITY with all access modes.	No action.
Opens for DENY NONE or DENY READ with READ access mode.	*F_RDLCK* only.
Opens for DENY NONE, DENY READ or DENY WRITE with WRITE and R/W access modes. In the case of DENY WRITE with R/W access, the record to be locked will be promoted to *F_WRLCK*. A record to be unlocked will be demoted to *F_RDLCK*.	*F_WRLCK* only.

Although LMX servers acquire an advisory lock prior to each READ or WRITE when interlocking is in effect, application developers should use byte-range locks whenever cooperating with CAE applications. This specification requires an LMX server to return an error if an access to a locked range takes place, which will cause many applications to fail.

Byte-range Locking

LMX servers must provide byte-range locking to SMB redirectors. There are some restrictions on the ability of an LMX server to completely emulate the required functionality of the SMB byte-range lock as it interacts with the access mode in which the file was opened. A file opened read-only access cannot have an *F_WRLCK* placed on it, as a CAE advisory write lock requires write permission. Because of this, an LMX server cannot simulate the SMB redirector R/W record locking semantics for read-only access.

Since the semantics of the SMB byte-range lock are mandatory rather than advisory, an LMX server must cause accesses by an SMB redirector to locked byte ranges to fail. Ideally, LMX servers would also cause access to those ranges from LMX server-resident processes to fail. This can only be accomplished if the LMX server-resident process is cooperative, that is, places advisory locks on byte ranges of interest, and if the LMX server places advisory locks on behalf of SMB redirector SMB requests.

The semantics of SMB locking require that an SMB redirector attempting to access (without locking) a range of bytes already locked by an LMX server-resident process must receive an error for that request. This means that an LMX server must place advisory locks for all SMB redirector SMB requests. These implicit locks exist solely for the time required for the requested operation and do not persist beyond that time. If an SMB redirector has already explicitly requested a lock, the LMX server need only maintain that lock and permit the SMB redirector to explicitly release it.

SMB byte-range locks can be larger than CAE file locks. The LMX server must support byte-range locks beyond standard CAE offsets.

4.4.2 Locking Timeouts

The extended dialect's requests for locking define timeout values that indicate how long the SMB redirector would like to wait before a lock attempt is failed. Support for these timeout values is not a requirement for an LMX server and may be ignored. If an LMX server cannot support timeouts, then the error <ERRSRV, ERRtimeout> is returned, just as if a timeout had occurred, if the resource is not available immediately upon request.

4.4.3 Read-only Locks

In the extended protocols, an LMX server may choose not to support read-only locks. It will then treat any request for such a lock as though a read/write lock has been requested.

4.5 LMX Server Caching

An LMX server may perform its own internal caching in an effort to increase performance for SMB redirectors. A simple example of this would be if the LMX server responds to write requests prior to making the CAE call necessary to write the data in the CAE system. This action by the LMX server is referred to as write-behind in the remainder of this document. By responding prior to writing the data, it means the SMB redirector may receive the response prior to the data being reflected in the CAE file system. If an LMX server does caching, it is required that it maintain this internal cache in such a manner that other SMB redirectors will see the same data if they make a read request prior to the CAE write by the server. It is not required that after an SMB redirector performs a write request, and receives the write response, that the data is reflected immediately to other CAE applications on the LMX server system. If an LMX server performs write-behind, it is required that the server honour *SMBflush* requests and not respond to these requests prior to flushing all appropriate, internally-cached data to the CAE file system.

4.6 LMX Print Spooling

The SMB protocols allow for status information on print jobs submitted to the LMX server. The LMX server, however, may choose to deal with print requests by a number of methods. One example would be for the LMX server to queue print requests internally to the server and then issue the requests to the CAE print spooling environment one job at a time, waiting for each job to complete before the next is spooled. This approach allows the LMX server to maintain state information concerning print requests that can be returned to the SMB redirector when necessary. Another approach is to couple the LMX server print queueing support with the CAE print spooling support. Depending on the degree the two are merged, it may not be possible for the LMX server to maintain the exact status of the print request, but a reasonable status must be estimated when necessary.

The print spooling protocols defined in Chapter 9 allow for the transmission of printer setup data, and give an indication of the type of data contained in the file (that is, text or graphics).

An LMX server implementation may choose to use or discard the printer setup data. The text or graphics mode indicator may be used by the LMX server to perform printer initialisation, or ignored.

4.7 SMB Error Codes

Chapter 5 defines a number of constants and descriptions of possible meanings for SMB error codes. In subsequent chapters, as each SMB is described, a table mapping possible error conditions to error codes is provided. If an LMX server implementation experiences an error condition that is not described in the table for the specific SMB, the LMX server may return any of the error codes defined in this document that best describe the error condition.

The ERRHRD class may cause an SMB redirector to notify the user of the error via an exception handling routine. Where the ERRHRD and ERRDOS class of errors overlap, the LMX server implementation has the option to use either class.

4.8 Security Policy

An LMX server must provide a security policy. It may provide either share-level security, user-level security, or a combination approach (refer to Section 2.2 on page 5 and Section 3.3 on page 12).

Another aspect of security is the support for encryption of user passwords. An LMX server may choose to support the encryption technique described in Appendix D or Section 11.2 on page 139. It is also acceptable for an LMX server not to support password encryption at all.

4.9 Negotiated Dialect

An LMX server may choose to support only one, a combination of, or all of the SMB dialects described in this document. Since the process of negotiating an SMB dialect is open-ended it is also possible that an LMX server supports dialects not described in this specification.

4.10 Network Issues

This specification assumes the LMX server implementation uses the transport support described in Appendix E on page 281 (TOP/NetBIOS), Appendix F on page 349 (RFC 1001) and Appendix G on page 419. It is for this reason that these RFCs are republished in this document.

For the binding of NetBIOS to the TCP/IP protocol suite (refer to Appendices F and G) only those aspects for B-node functionality are required.

An implementation may choose to support the full M-node functionality, as that is a superset of B-node.

For the binding of NetBIOS to OSI transport (refer to Appendix E on page 281) the NetBIOS user agent is optional.

This specification defines a default method by which LMX servernames are mapped to NetBIOS names (refer to Section 3.5.2 on page 15). It is possible that an LMX server implementation and compatible SMB redirector implementation may use additional methods of mapping LMX servernames to NetBIOS names.

SMB protocols are only specified to run on a single LAN subnetwork, but interoperation in connected subnetworks is not precluded.

X/Open has defined other types of PC connectivity support; refer to the X/Open Developers' Specification, Protocols for X/Open PC Interworking: (PC)NFS. (PC)NFS and SMB protocol implementations, or other connectivity implementations, on the same server are not required to interwork with respect to additional features beyond those provided by XSI (for example, extended DOS file open modes). Additionally, if the CAE system is supporting access to other CAE systems via XNFS (reference X/Open CAE Specification, Protocols for X/Open Interworking: XNFS), it may be possible to configure an LMX server to allow SMB redirectors access to the resources of the other CAE systems via the XNFS connection, but this is not a requirement.

Chapter 5

Data Objects and Constants

This chapter describes the SMB format, common data structures, flag fields and other objects commonly used in SMB requests and responses. It also defines various symbolic constants and indicates their (required) values. Throughout the specification the following definitions will be used:

8-bit field	An octet; sometimes referred to as a byte.
16-bit	Two 8-bit fields with the least significant 8-bit field first (little-endian).
32-bit	Two 16-bit elements with the least significant 16-bit element first (little-endian).

5.1 SMB Format

All SMB requests and responses (except where noted) have a common header, as follows:

Offset	Type	Field Name	Description
00	8-bit field	*smb_idf*[4]	contains 0xff,0x53,0x4d,0x42
04	8-bit field	*smb_com*	command code
05	8-bit field	*smb_rcls*	error class
06	8-bit field	*smb_reh*	reserved for future
07	16-bit field	*smb_err*	error code
09	8-bit field	*smb_flg*	flags
10	16-bit field	*smb_res*[7]	reserved for future
24	16-bit field	*smb_tid*	authenticated resource identifier
26	16-bit field	*smb_pid*	caller's process ID
28	16-bit field	*smb_uid*	unauthenticated user ID
30	16-bit field	*smb_mid*	multiplex ID
32	8-bit field	*smb_wct*	count of 16-bit fields that follow
33	16-bit field	*smb_vwv*[]	variable number of 16-bit fields
-	16-bit field	*smb_bcc*	count of 8-bit fields that follow
-	8-bit field	*smb_buf*[]	variable number of 8-bit fields

The structure defined from *smb_idf* through *smb_wct* is the fixed portion of the SMB structure sometimes referred to as the SMB header. Following the header there is a variable number of 16-bit fields (defined by *smb_wct*), and following that is *smb_bcc* which defines an additional variable number of 8-bit fields. The SMB header fields are defined as follows:

smb_idf SMB identification string, always 0xff,0x53,0x4d,0x42

smb_com SMB command code (see Section 5.2 on page 40).

smb_rcls Error class (see Section 5.6 on page 49), set in the SMB response only.

smb_err Error code (see Section 5.6 on page 49), set in the SMB response only.

smb_flg A bit-encoded field. The flag bits are defined as follows:

Bit 0 When set (returned) by the LMX server in the *SMBnegprot* response, this bit indicates that the LMX server supports the *SMBlockread* and *SMBwriteunlock* requests.

Bit 1 Used only in requests when an extended SMB protocol is negotiated. When set, the SMB redirector guarantees a receive buffer is already posted; this has implications for the type of underlying transport service which may be used in sending a response.

Bit 2 Reserved; MBZ (Must Be Zero).

Bit 3 When on, all pathnames in the protocol must be treated as case-insensitive. If one of the extended protocols is negotiated and the bit is set off, the pathnames are case-sensitive. The LMX server can assume the value is always set to on.

Bit 4 Used only in the *SMBsesssetupX* request. When on, the SMB redirector indicates that all pathnames will be specified as canonical pathnames, already obeying the file naming conventions (see Section 3.5 on page 15). When off, pathnames are in the LMX server representation. The LMX server can assume the value is always set to on.

Bit 5 Used only in the *SMBopen*, *SMBcreate* and *SMBmknew* requests/responses. When set in a request, the SMB redirector asks that the file be opportunistically locked, a feature of the extended SMB protocols. If the LMX server places the opportunistic lock, this bit is set in the SMB response. This bit is referred to as the oplock bit.

Bit 6 Used only in the *SMBopen*, *SMBcreate* and *SMBmknew* requests when an extended protocol is negotiated; meaningful only if bit 5 is also set. When set, the SMB redirector is asking to be notified of any operation which can modify the file (for example, delete, setting of attributes, rename, etc.). This allows the redirector to cache the complete file. If not set, the SMB redirector need only be notified if another open request is received for the file. This bit is referred to as the opbatch bit.

Bit 7 Always set in responses. The *smb_com* (command code) field usually contains the same value in a request from the SMB redirector to the LMX server as in the matching SMB response from the LMX server to the SMB redirector. This bit unambiguously distinguishes the SMB request from the SMB response. On a multiplexed LMX session on a system where both LMX server and SMB redirector are active, this bit can be used by the system's SMB delivery system to help identify whether this protocol should be routed to a waiting SMB redirector or to the LMX server.

smb_tid Used by the LMX server to identify a resource (for example, a file system subtree). The value 0xffff is reserved. The LMX server is responsible for enforcing use of a valid TID where appropriate (see Section 3.2 on page 10).

smb_pid Generated by the SMB redirector to uniquely identify a process within the SMB redirector's system. An SMB response will always contain the same value in *smb_pid* (and *smb_mid*) as in the corresponding SMB request.

smb_uid User identifier. It is used by the extended protocol when the LMX server is executing in user-level security mode to validate access on requests which reference named resources (such as file open). Refer to Section 3.2 on page 10, Section 3.3 on page 12 and Section 4.3.1 on page 30 for additional information. Thus differing users accessing the same TID may be granted differing access to

the resources defined by the TID based on *smb_uid*. The username and password requested are validated by the LMX server via the *SMBsesssetupX* exchange (refer to Section 11.3 on page 144). The LMX server returns a value in *smb_uid* that will be used by the SMB redirector to represent the user identity requested.

Note that 0xfffe (–2) is reserved as an invalid UID. In share-level security mode this field is not used.

smb_mid This field is used for multiplexing multiple SMBs on a single LMX session. The PID (in *smb_pid*) and the MID (in *smb_mid*) uniquely identify a request and are used by the SMB redirector to correlate incoming SMB responses to previously sent SMB requests (refer to Section 3.2 on page 10).

5.2 SMB Command Codes

This table shows the mapping between the symbolic name for an SMB request or response and the value to be placed in the *smb_com* field of the SMB header. The Protocol column indicates the protocol class to which the request belongs:

C Core protocol; all dialects.

C+ Core plus protocol as generated by the 1.03 dialect.

E Extended protocol; only those dialects defined as extended 1.0.

E2 Extended protocol; only those dialects defined as extended 2.0.

- Not generated by dialects of LAN Manager; included for reference purposes only.

Name	*smb_com*	Protocol	
SMBmkdir	0x00	C	
SMBrmdir	0x01	C	
SMBopen	0x02	C	
SMBcreate	0x03	C	
SMBclose	0x04	C	
SMBflush	0x05	C	
SMBunlink	0x06	C	
SMBmv	0x07	C	
SMBgetatr	0x08	C	
SMBsetatr	0x09	C	
SMBread	0x0a	C	
SMBwrite	0x0b	C	
SMBlock	0x0c	C	
SMBunlock	0x0d	C	
SMBctemp	0x0e	Reserved	
SMBmknew	0x0f	C	
SMBchkpth	0x10	C	
SMBexit	0x11	C	
SMBlseek	0x12	C	
SMBlockread	0x13	C+	
SMBwriteunlock	0x14	C+	
SMBreadbraw	0x1a	C+	
SMBreadbmpx	0x1b	E	
SMBreadbs	0x1c	E	
SMBwritebraw	0x1d	C+	
SMBwritebmpx	0x1e	E	
SMBwritebs	0x1f	E	
SMBwritec	0x20	E	
reserved	0x21	-	
SMBsetattrE	0x22	E	
SMBgetattrE	0x23	E	
SMBlockingX	0x24	E	
SMBtrans	0x25	E	See Note.
SMBtranss	0x26	E	

Name	*smb_com*	Protocol	
SMBioctl	0x27	E	
SMBioctls	0x28	E	
SMBcopy	0x29	E	
SMBmove	0x2a	E	
SMBecho	0x2b	E	
SMBwriteclose	0x2c	E	
SMBopenX	0x2d	E	
SMBreadX	0x2e	E	
SMBwriteX	0x2f	E	
SMBtrans2	0x32	E2	
SMBtranss2	0x33	E2	
SMBfindclose	0x34	E2	
SMBfindnclose	0x35	E2	
SMBlogon	0x60	-	
SMBbind	0x61	-	
SMBunbind	0x62	-	
SMBgetaccess	0x63	-	
SMBlink	0x64	-	
SMBfork	0x65	-	Reserved for
SMBgetpath	0x68	-	proprietary
SMBreadh	0x69	-	dialects
SMBrdchk	0x6b	-	
SMBmknod	0x6c	-	
SMBrlink	0x6d	-	
SMBgetlatr	0x6e	-	
SMBtcon	0x70	C	
SMBtdis	0x71	C	
SMBnegprot	0x72	C	
SMBsesssetupX	0x73	E	
SMBulogoffX	0x74	E2	
SMBtconX	0x75	E	
SMBdskattr	0x80	C	
SMBsearch	0x81	C	
SMBffirst	0x82	E	
SMBfunique	0x83	E	
SMBfclose	0x84	E	
SMBsplopen	0xc0	C	
SMBsplwr	0xc1	C	
SMBsplclose	0xc2	C	
SMBsplretq	0xc3	C	
SMBsends	0xd0	C	
SMBsendb	0xd1	C	
SMBfwdname	0xd2	C	
SMBcancelf	0xd3	C	
SMBgetmac	0xd4	C	
SMBsendstrt	0xd5	C	
SMBsendend	0xd6	C	

Name	*smb_com*	Protocol
SMBsendtxt	0xd7	C
Never valid	0xfe	Never sent
Implementation-dependent	0xff	-

Note: The *SMBtrans* request is used within the extended SMB protocols only for services described in the X/Open CAE Specification, IPC Mechanisms for SMB and is outside the scope of this specification.

5.3 Data Objects

This section defines various fields, objects and structures used in more than one SMB request or response.

5.3.1 Time Fields

There are two time field formats; one 16 bits in length, and one 32 bits in length. Many SMBs contain a 16-bit quantity whose value indicates a particular time. Unless otherwise specified, the time is encoded in the following format:

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
h	*h*	*h*	*h*	*h*	*m*	*m*	*m*	*m*	*m*	*m*	*x*	*x*	*x*	*x*	*x*

hhhhh Bits 11-15 contain the current hour; range is 0-23.

mmmmmm Bits 5-10 contain the current minute; range is 0-59.

xxxxx Bits 0-4 contain the current seconds in units of two seconds; range is 0-29.

Other SMBs contain a 32-bit value which represents a time, in seconds, relative to midnight on January 1, 1970 (the Epoch). This 32-bit value is a signed, but always positive, 32-bit integer, and is split into two 16-bit values in the SMB. The low-order 16-bit values are always first, followed immediately by the high-order 16-bit values. This pair is usually referred to as time low and time high.

5.3.2 Date Fields

Many SMBs contain a 16-bit value indicating a particular date. Unless otherwise specified, the date is encoded in the following format:

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
y	*y*	*y*	*y*	*y*	*y*	*y*	*m*	*m*	*m*	*m*	*d*	*d*	*d*	*d*	*d*

yyyyyyy Bits 9-15 contain the current year, less 1980; range is 0-119, indicating 1980-2099. Note that the base year is not 1970.

mmmm Bits 5-8 contain the current month; range 1-12, where 1 is January.

ddddd Bits 0-4 contain the current day of the month; range 1-31.

5.3.3 File Attributes Fields

Many SMBs contain one or more 16-bit values, each of which encodes file attributes. Unless otherwise specified, the attributes are encoded in the following format:

Bit 0 The file is read-only.

Bit 1 The file is hidden.

Bit 2 The file is a system file.

Bit 3 The file is a volume identifier.

Bit 4 The file is a directory.

Bit 5 The file is flagged as changed since last archive.

All other bits are reserved and Must Be Zero. If none of the attribute bytes are set, the file attributes refer to a regular file. Note that use of this field is governed by the File Attributes conventions (see Section 4.3.1 on page 30).

5.3.4 Buffers

Many of the core SMBs contain typed buffers in the *smb_buf* field. A buffer consists of a single 8 bit field, indicating the type of buffer, followed by a string of 8-bit fields, which are the contents of the buffer. The buffer type defines the termination method for the buffer contents. The buffer types are:

01 Data Block. The buffer contains a 16-bit value containing the length of the data block, followed by that number of 8 bit fields of data. This buffer is not null-terminated.

02 Dialect. The buffer is a null-terminated string of bytes making up a dialect name (see Section 5.4 on page 48).

04 ASCIIZ. The buffer is a null-terminated string of ASCII characters.

05 Variable Block. The buffer contains a 16-bit value containing the length of the data block, followed by that number of 8 bit fields of data. This buffer is not null-terminated.

5.3.5 File-sharing Control

SMBs which open files make use of a 16-bit value to control the extent of file sharing to be permitted. This 16-bit value has the following format:

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	w	0	0	0	0	0	0	r	*x*	*x*	*x*	*y*	*y*	*y*	*y*

Bits 8-13 and bit 15 are reserved and should be ignored by the LMX server.

w Write-through mode. Neither read-ahead nor write-behind caching for this file is permitted. An LMX server should not respond to any SMB request involving this file until all data related to the SMB request is on stable store (that is, on disk). This mode is generated in extended protocols only.

r Reserved. Ignored by the LMX server.

xxx Exclusion mode. Values are:

- 0 DOS compatibility mode (exclusive to an LMX session, but that LMX session may have multiple opens).
- 1 DENY ALL (exclusive to this operation).
- 2 DENY WRITE. Other users may access the file in READ mode. Open for executing is not allowed.
- 3 DENY READ. Other users may access the file in WRITE mode.
- 4 DENY NONE. Allow other users to access the file in any mode for which they have permission.

5,6 Illegal. SMB redirectors should not specify these values.

7 FCB open mode (see below).

yyyy Type of access requested. Values are:

0 Open the file for reading.

1 Open the file for writing.

2 Open the file for reading and writing.

3 Open the file for executing (extended protocols only).

4 14 Illegal. SMB redirectors should not specify these values.

15 Illegal, except for FCB open (see below).

For the exclusion modes see Section 3.7.2on page 18.

Special semantics, called an FCB open, are associated with a file-sharing control value of 0x00ff. This type of open will cause a DOS compatibility open with the read/write modes set to the maximum permissible. Generally, this will cause any access violations to be detected when the first read and/or write is attempted, rather than during open processing.

The open for execute bit maps to read-only, and writes to these files from SMB redirectors are not allowed while that attribute is set.

5.3.6 Resource Types

In *SMBtcon* and *SMBtconX* an ASCIIZ buffer (type 04) is used to specify the resource type. The following are acceptable:

A: File system share.

LPT 1: Spoolable device.

COMM Character mode device.

IPC $ Mailslots or named pipes.

SMBopenX contains a 16-bit field denoting a resource type. The permissible values for this field are:

0 File or directory, as determined by the attribute field *smb_attr* related to the same file.

1 Stream mode named pipe - see the X/Open CAE Specification, IPC Mechanisms for SMB.

2 Message mode named pipe - see the X/Open CAE Specification, IPC Mechanisms for SMB.

3 Printer device.

4 Character mode device. When an extended protocol has been negotiated, it allows a device to be opened (via *SMBopenX*) for driver-level I/O. This provides direct access to real-time and interactive devices such as modems, scanners, etc.

Named Pipes, Mailslots and Messaging

Named pipes, mailslots and messaging are IPC mechanisms defined in the X/Open CAE Specification, IPC Mechanisms for SMB which are outside the scope of this specification. To support named pipes and mailslots extended SMB protocol elements are required that will use specific resource types as defined above. Two such types of devices are defined:

COMM Communication devices like modems or terminals.

LPT1 Printer devices which will be accessed directly.

5.3.7 Access Modes

Some SMBs which open files return an indication of the type of access granted to the requestor. This 16-bit field takes the following values:

0 Read-only access granted.

1 Write-only access granted.

2 Read/write access granted.

≥3 Reserved; do not use.

5.3.8 Open Function

The open function field controls the way a file should be treated when it is opened for use by certain extended SMB requests. This 16-bit field is bit-encoded:

Bits 0-1 This field determines the action to be taken if the file exists. The values and meanings for this field are:

- 0 The request should fail and an error returned indicating the prior existence of the file.
- 1 The file should be appended to.
- 2 The file should be truncated to zero (0) length.
- 3 Reserved; this value should not be used.

Bit 4 If the file does not exist and this bit is clear, the request should fail; if this bit is set, the file should be created.

All other bits are reserved and should be ignored by the LMX server.

5.3.9 Resource Names, Pathnames, Filenames and Network Pathnames

A pathname is a 1 to 255 byte long UNC name that routes to a directory.

A filename is an 8.3 format or long filename format name that routes to a file. In the case of the extended 2.0 dialect a filename may be up to 255 bytes in length. A pathname may be included to specify a directory where the file resides.

A network pathname is a filename proceeded by the LMX servername and has the following format:

\\<LMXservername>\<pathname>\<filename>

where:

<LMXservername> is a one to fifteen byte LMX servername.

<pathname> is a collection of component names in either the 8.3 format or in a long filename format.

<filename> is the final 8.3 or long filename format name.

5.3.10 File Identifiers

Many SMB requests and responses contain a 16-bit file identifier (FID). These are created by the LMX server upon an open request and need to be maintained by the SMB redirector. All values but –1 (0xFFFF) are valid. The –1 is used to specify all FIDs or no FID, depending on the context by which it is used.

5.4 SMB Dialects

To distinguish between various levels of SMB protocols the SMB redirector will send in the *SMBnegprot* request (see Section 6.1 on page 56) a set of dialect strings from which the LMX server will select one to be used for the LMX session. The currently known dialect strings are:

Dialect String	Referred to as
PC NETWORK PROGRAM 1.0	core protocol
MICROSOFT NETWORKS 1.03	core plus dialect
MICROSOFT NETWORKS 3.0	extended 1.0 protocol
LANMAN 1.0	extended 1.0 protocol
LM1.2X002	extended 2.0 protocol

MICROSOFT NETWORKS 3.0 and LANMAN 1.0 specify the same SMB protocol dialect. MICROSOFT NETWORKS 3.0 is used by DOS SMB redirectors and LANMAN 1.0 is used by OS/2 SMB redirectors. The MICROSOFT NETWORKS 1.03 string specifies a slightly extended version of the core protocol. The LM1.2X002 protocol specifies the second extension to the protocols. This dialect is used to provide longer names to files and other file characteristics to the SMB environment.

5.5 Timeouts

Some of the SMB protocols allow for the operation to time out prior to its success or failure. This timeout feature allows SMB redirectors to attempt to open devices which may not open immediately. For example, an application that requires the services of a modem may be running on the SMB redirector system. An LMX server may provide a modem pool and allow SMB redirector access to this modem pool. When the SMB redirector attempts to open a modem device, the open request may be queued until a modem is free. By specifying a timeout on the open request, the SMB redirector is able to return a busy error to the user of the modem application when all of the modems are busy rather than wait indefinitely.

Timeout values within the SMB protocol are typically 32-bit values representing the number of milliseconds the SMB redirector would like before the request is returned with an error (exceptions are noted in the text when a timeout is defined). Some timeout values are reserved for the following function:

0 Return immediately if the request cannot be satisfied at this time.

–1 Wait indefinitely.

–2 Wait for an LMX server-defined default. This default time is implementation-dependent. Suggested defaults depend on the type of activity requested. For example, writes may have an infinite timeout, but opens may have a timeout in the range of 10 to 20 seconds.

5.6 SMB Error Codes

This section specifies the error class and error code values for the SMB headers. In SMB responses the error class will be set in the SMB header field *smb_rcls*. The error code will be set in the SMB header field *smb_err*. If a value is not listed it is considered reserved for future use. Some of the error codes will only occur when SMBs are used to implement the X/Open CAE Specification, IPC Mechanisms for SMB, which is outside the scope of this specification.

In the case of success, the LMX server must return error class SUCCESS and error code SUCCESS. An undefined error (for example, caused by a corrupted SMB, internal LMX server error) should be in error class ERRSRV and error code ERRerror.

5.6.1 SMB Error Class Mappings

Unless otherwise stated, the following error classes may be returned.

Name	Value	Description
SUCCESS	0x00	The request was successful.
ERRDOS	0x01	Error is considered to be operating system related.
ERRSRV	0x02	Error is generated by the LMX server.
ERRHRD	0x03	Error is a hardware error.
ERRXOS	0x04	Reserved.
ERRRMX1	0xe1	Reserved.
ERRRMX2	0xe2	Reserved.
ERRRMX3	0xe3	Reserved.
ERRCMD	0xff	Command was not in the SMB format.

The ERRXOS, ERRRMX1, ERRRMX2 and ERRRMX3 error classes are not used in the SMB protocols defined in this specification.

5.6.2 Error Codes for the SUCCESS Class

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

Name	Value	Description
SUCCESS	0x00	The request was successful.
BUFFERED	0x54	Message was buffered (used in Messaging).
LOGGED	0x55	Message was logged (used in Messaging).
DISPLAYED	0x56	Message was displayed (used in Messaging).

Note: Messaging is described in the X/Open CAE Specification, IPC Mechanisms for SMB and is outside the scope of this specification.

5.6.3 Error Codes for the ERRDOS Class

In general, the ERRDOS class is used to return OS-specific errors to SMB redirectors. Since the SMB redirector needs to understand these error codes for all LMX servers, it is impossible to define CAE-specific errors. Instead, the list of possible error codes, with some explanatory text, appears below. An LMX server may elect to return one of these more specific error codes any time a system-specific error occurs.

The Name column gives the symbolic name for the error. The Value column indicates the numeric value for the constant, and a description follows in the Description column. A hint to the CAE error code (see Chapter 23, Error Numbers, of the X/Open Portability Guide, Issue 3, Volume 2, XSI System Interface and Headers) that may be mapped to the SMB error code is given in the description text.

Name	Value	Description
ERRbadfunc	1	Invalid function. The LMX server's OS did not recognise or could not perform a system call generated by the LMX server; for example, set the directory file attribute on a data file, invalid seek mode. [EINVAL]
ERRbadfile	2	File not found. The last component of a file's pathname could not be found. [ENOENT]
ERRbadpath	3	Directory invalid. A directory component in a pathname could not be found. [ENOENT]
ERRnofids	4	Too many open files. The LMX server has no FIDs available. [EMFILE]
ERRnoaccess	5	Access denied, the requestor's context does not permit the requested function. This includes the following conditions: invalid rename command, write to FID open for read-only, read on FID open for write-only, attempt to delete a non-empty directory. [EPERM]
ERRbadfid	6	Invalid FID. The FID specified was not recognised by the LMX server. [EBADF]
ERRnomem	8	Insufficient LMX server memory to perform the requested function. [ENOMEM]
ERRbadmem	9	Invalid memory block address. [EFAULT]
ERRbadenv	10	Invalid environment.
ERRbadaccess	12	Invalid open mode.
ERRbaddata	13	Invalid data (generated only by IOCTL calls within the LMX server). [E2BIG]
ERRres	14	Reserved.
ERRbaddrive	15	Invalid drive specified. [ENXIO]
ERRremcd	16	A Delete Directory request attempted to remove the LMX server's current directory.
ERRdiffdevice	17	Not the same device (for example, a rename across different file systems was attempted). [EXDEV]
ERRnofiles	18	A File Search command can find no more files matching the specified criteria.
ERRbadshare	32	The sharing mode specified for an Open conflicts with existing FID on the file. [ETXTBSY]
ERRlock	33	A Lock request conflicted with an existing lock or specified an invalid mode, or an Unlock request attempted to remove a lock held by another process. [EDEADLOCK]
ERRfilexists	80	The file named in a Create Directory, Make New File or Link request already exists. The error may also be generated in the Create and Rename transaction. [EEXIST]

Name	Value	Description
ERRbadpipe	230	Named pipe invalid.
ERRpipebusy	231	All instances of the requested pipe are busy.
ERRpipeclosing	232	Named pipe close in progress.
ERRnotconnected	233	No process on the other end of the named pipe.
ERRmoredata	234	There is more data to be returned.
ERROR_EAS_DIDNT_FIT	275	There are no extended attributes, or the number of attributes available did not fit into the SMB response.
ERROR_EAS_NOT_SUPPORTED	282	The LMX server does not support storage of extended attributes.

5.6.4 Error Codes for the ERRSRV Class

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

Name	Value	Description
ERRerror	1	Non-specific error code. It is returned under the following conditions: resource other than file system space exhausted (for example,TIDs), first command on the LMX session was not *SMBnegprot*, multiple *SMBnegprots* attempted, or internal LMX server error.
ERRbadpw	2	Bad password - name/password pair in an *SMBtcon*, *SMBtconX* or *SMBsesssetupX* are invalid.
ERRbadtype	3	Reserved.
ERRaccess	4	The requestor does not have the necessary access rights within the specified context for the requested function. The context is defined by the TID or the UID. [EACCES]
ERRinvnid	5	The TID specified in a command was invalid.
ERRinvnetname	6	Invalid LMX servername in *SMBtcon* or *SMBtconX*
ERRinvdevice	7	Invalid device - printer request made to non-printer connection or non-printer request made to printer connection.
ERRqfull	49	Print queue full (that is, too many queue items) - returned by open print file.
ERRqtoobig	50	Print queue full (that is, no space or queued item too big).
ERRinvpfid	52	Invalid print file specified in *smb_fid*.
ERRsmbcmd	64	The LMX server did not recognise the command code received.
ERRsrverror	65	The LMX server encountered an internal error.
ERRfilespecs	67	The FID and pathname parameters contained an invalid combination of values.
ERRbadlink	68	Reserved.
ERRbadpermits	69	The access permissions specified for a file or directory are not a valid combination. The LMX server cannot set the requested attribute.

Name	Value	Description
ERRbadpid	70	Reserved.
ERRsetattrmode	71	The attribute mode in the Set File Attribute request is invalid.
ERRpaused	81	Message server is paused. (Reserved for messaging.)
ERRmsgoff	82	Not receiving messages. (Reserved for messaging.)
ERRnoroom	83	No room to buffer message. (Reserved for messaging.)
ERRtmuns	87	Too many remote usernames. (Reserved for messaging.)
ERRtimeout	88	Operation timed out.
ERRnoresource	89	No resources currently available for SMB request.
ERRtoomanyuids	90	Too many UIDs active on this LMX session.
ERRbaduid	91	The UID given (*smb_uid*) is not known as a valid ID on this LMX session.
ERRuseMPX	250	Temporarily unable to support Raw mode operation, use MPX mode.
ERRuseSTD	251	Temporarily unable to support Raw mode operation, use standard read/write.
ERRcontMPX	252	Continue in MPX mode.
ERRBadPW	254	Reserved.
ERRnosupport	0xffff	Function not supported.

5.6.5 Error Codes for the ERRHRD Class

The following error codes may be generated for hard errors on the LMX server with the ERRHRD error class. CAE error mapping hints to each of these errors are noted at the end of the error description.

The ERRHRD error class may cause an SMB redirector to notify the user of the error condition via an exception handling routine. Where ERRHRD and ERRDOS error classes overlap, the LMX server implementation has the option to choose an appropriate class for the error.

Name	Value	Description
ERRnowrite	19	Attempt to write on write-protected diskette. [EROFS]
ERRbadunit	20	Unknown unit. [ENODEV]
ERRnotready	21	Drive not ready. [EUCLEAN]
ERRbadcmd	22	Unknown command.
ERRdata	23	Data error (CRC). [EIO]
ERRbadreq	24	Bad request structure length. [ERANGE]
ERRseek	25	Seek error.
ERRbadmedia	26	Unknown media type.
ERRbadsector	27	Sector not found.
ERRnopaper	28	Printer out of paper.
ERRwrite	29	Write fault.
ERRread	30	Read fault.
ERRgeneral	31	General hardware failure.
ERRbadshare	32	An open conflicts with an existing open. [ETXTBSY]
ERRlock	33	A Lock request conflicted with an existing lock or specified an invalid mode, or an Unlock request attempted to remove a lock held by another process. [EDEADLOCK]

Name	Value	Description
ERRwrongdisk	34	The wrong disk was found in a drive.
ERRFCBUnavail	35	No FCBs are available to process the request.
ERRsharebufexc	36	A sharing buffer has been exceeded.
ERRdiskfull	39	No space on file system. [ENOSPC]

Chapter 6

Core SMB Connection Management Requests

This section defines the elements of the core SMB protocol related to connection management. They are:

SMBnegprot	negotiate protocol
SMBtcon	tree connect
SMBtdis	tree disconnect
SMBexit	process exit

6.1 SMBnegprot Specification

SMBnegprot Detailed Description

This core protocol request is sent as the first request to establish the LMX session, negotiating the protocol dialect that the SMB redirector and LMX server will use when communicating with each other. The SMB redirector sends a list of dialects that he can communicate with. The LMX server responds with a selection of one of those dialects (numbered 0 to *n*) or – 1 indicating that none of the dialects were acceptable. Exactly one negotiate message must be sent on each NetBIOS session; subsequent negotiate requests must be rejected with an error response and no action will be taken.

The SMB protocol does not impose any particular structure on the dialect strings. Implementors of particular protocols may choose to include, for example, version numbers in the string. An LMX server may choose to support one or more of the dialects identified in Section 5.4 on page 48. The fields described here are only valid when the core protocol has been negotiated. The other SMB dialects impose some differences on the *SMBnegprot* format; refer to the sections discussing the different dialects for information on these differences.

SMBnegprot Deviations

None.

SMBnegprot Field Descriptions

Field descriptions for the core protocol (*SMBnegprot*) are as follows:

From SMB redirector		To SMB redirector	
Field Name	Field Value	Field Name	Field Value
smb_com	*SMBnegprot*	*smb_com*	*SMBnegprot*
smb_wct	0	*smb_wct*	1
smb_bcc	min = 2	*smb_vwv*[0]	*smb_index*
smb_buf[ ]	dialect0	*smb_bcc*	0
	.		
	.		
	dialect*n*		

SMBnegprot Error Code Descriptions

If any error occurs, the server will return <ERRSRV, ERRerror>; otherwise, <SUCCESS, SUCCESS> will be returned.

SMBnegprot Preconditions

The SMB redirector attempting to negotiate a protocol must have established a NetBIOS session with the server.

SMBnegprot Postconditions

The SMB redirector that negotiated this protocol must be able to handle all aspects of the dialect negotiated.

SMBnegprot Side Effects

The LMX server will keep record of which dialect the SMB redirector negotiated and will use only that dialect in conversations with the SMB redirector.

Coventions

None.

6.2 SMBtcon Specification

SMBtcon Detailed Description

This core protocol request is sent to establish direct access to a resource on an LMX server. The exact behaviour of this request and the semantics of the password argument depend upon the security mode of the LMX server.

- share-level security mode

  The password establishes the user's rights to access this resource. It must match the password (if any) defined by the server administrator when the resource was made available for sharing (offered).

- user-level security mode

  Based on the negotiated dialect, an LMX server in user-level security must behave in one of two different ways:

  — If one of the extended SMB protocol dialects was selected the SMB redirector has already issued an *SMBsesssetupX* request. This request contained a username and password and resulted in the LMX server assigning a valid UID (refer to Section 3.3.2 on page 12). In this case, the password field will be meaningless and must be ignored.

  — If the core or core plus dialect was selected, the SMB redirector will issue an *SMBtcon* request as if the LMX server were in share-level security mode. The LMX server may select to support a mapping to user-level security (refer to Section 3.3.3 on page 13). The password supplied with the *SMBtcon* request can be used for this validation.

SMBtcon Deviations

None.

SMBtcon Field Descriptions

From SMB redirector		To SMB redirector	
Field Name	Field Value	Field Name	Field Value
smb_com	*SMBtcon*	*smb_com*	*SMBtcon*
smb_wct	0	*smb_wct*	2
smb_bcc	min=4	*smb_vwv*[0]	*smb_maxxmt*
smb_buf[]	*smb_path*	*smb_vwv*[1]	TID
	smb_password	*smb_bcc*	0
	smb_device		

smb_path An ASCIIZ buffer (type 04; refer to Section 5.3.4 on page 44) containing a resource name preceded by the LMX servername. The format is like a network pathname (refer to Section 5.3.9 on page 46). For example, a resource called src residing on a server called lmserver1 would be referenced by \\lmserver1\src.

smb_password An ASCIIZ (type 04) buffer containing the password for the resource. Total length of the buffer must be less than or equal to 15 bytes. For the extended protocols the encrypted password string can be up to 24 bytes.

smb_device An ASCIIZ (type 04) buffer containing the resource type. Refer to Section 5.3.6 on page 45.

smb_maxxmt A 16 bit integer defining the largest message that the SMB redirector can send to the LMX server and *vice versa*.

TID (Tree ID) A 16 bit integer used by the LMX server in subsequent SMB redirector requests to refer to a resource relative to *smb_path*. Most access to the server requires a valid TID, whether the resource is password protected or not. The *smb_tid* field in the SMB header of this request is ignored. The value 0xffff is reserved.

SMBtcon Error Code Descriptions

CAE Code	DOS Class	DOS Code	Description
-	SUCCESS	SUCCESS	Everything worked, no problems.
-	ERRDOS	ERRnomem	A memory related resource has depleted.
-	ERRDOS	ERRbadpath	The CAE path related to the resource is not valid.
-	ERRSRV	ERRinvdevice	Resource type mismatch for connect.
-	ERRSRV	ERRaccess	User not authorised to access specified resource.
-	ERRSRV	ERRerror	Ran out of TIDs.
-	ERRSRV	ERRerror	First command on the NetBIOS session wasn't *SMBnegprot*.
-	ERRSRV	ERRerror	LMX server internal error.
-	ERRSRV	ERRbadpw	Bad password, name/password pair in an *SMBtcon* is invalid.
-	ERRSRV	ERRinvnetname	Invalid resource name supplied in the *SMBtcon*.

SMBtcon Preconditions

1. The SMB redirector attempting to set up this *SMBtcon* must have established an LMX session with the LMX server.
2. The path, password and device name must all be valid instances of those types.

SMBtcon Postconditions

1. If there are no errors the TID is valid to be used in future SMB requests until it is nullified with an *SMBtdis* request. Otherwise, the TID should not be used in future transactions.
2. If there are no errors the *smb_maxxmt* size will represent the negotiated maximum buffer size for the LMX session.

SMBtcon Side Effects

None.

Conventions

- Resource Names (see Section 5.3.9 on page 46) applies to the *smb_path* field.

6.3 SMBtdis Specification

SMBtdis Detailed Description

This core protocol request is sent to invalidate the resource (file or print) sharing connection identified by the TID.

SMBtdis Deviations

None.

SMBtdis Field Descriptions

From SMB redirector		To SMB redirector	
Field Name	Field Value	Field Name	Field Value
smb_com	*SMBtdis*	*smb_com*	*SMBtdis*
smb_wct	0	*smb_wct*	0
smb_bcc	0	*smb_bcc*	0

There are no parameters of interest besides the TID (passed in the *smb_tid* field of the SMB header). If an invalid TID is sent, the server will ignore the request and return an error.

SMBtdis Error Code Descriptions

CAE Code	DOS Class	DOS Code	Description
-	SUCCESS	SUCCESS	Everything worked, no problems.
-	ERRSRV	ERRbaduid	The UID given (*smb_uid*) is not known as a valid ID on this LMX session.
-	ERRSRV	ERRinvnid	TID specified in command was invalid.
-	ERRSRV	ERRerror	LMX server internal error.

SMBtdis Preconditions

1. The SMB redirector attempting to invalidate this TID must have established an LMX session with the LMX server.
2. The SMB redirector attempting to invalidate this TID should have established this TID as a valid one with the LMX server.

SMBtdis Postconditions

1. If there are no errors then the TID will be invalidated and the SMB redirector should not use the TID again.
2. If an error other than TID Invalid occurs, the TID will be invalidated and the SMB redirector should not use the TID again.

SMBtdis Side Effects

The TID that was sent no longer has any meaning to the LMX server.

Conventions

None.

6.4 SMBexit Specification

SMBexit Detailed Description

This core protocol request informs the LMX server that an SMB redirector process has terminated.

The LMX server will release any locks and close any resources owned by the exiting process.

Note that there is no process creation SMB request. PIDs are assigned by the SMB redirector.

SMBexit Deviations

An LMX server should accept this request from any LMX session regardless of dialect.

SMBexit Field Descriptions

From SMB redirector		To SMB redirector	
Field Name	Field Value	Field Name	Field Value
smb_com	*SMBexit*	*smb_com*	*SMBexit*
smb_wct	0	*smb_wct*	0
smb_bcc	0	*smb_bcc*	0

The *smb_pid* field from the SMB header indicates the process to be terminated.

SMBexit Error Code Descriptions

CAE Code	DOS Class	DOS Code	Description
-	SUCCESS	SUCCESS	Everything worked, no problems.
-	ERRSRV	ERRinvnid	Bad TID.
-	ERRSRV	ERRerror	Some other error occurred.

SMBexit Preconditions

The SMB redirector must have registered a UID and established a TID with the LMX server.

SMBexit Postconditions

None.

SMBexit Side Effects

None.

Conventions

None.

Chapter 7

Core SMB File Operation Requests

This section defines the elements of the core SMB protocol related to normal file access. They are:

SMBcreate	open a file; create it if it doesn't exist
SMBmknew	create and open a new file; fail if it exists
SMBopen	open an existing file
SMBread	read from a file
SMBwrite	write to a file
SMBlseek	set the current position in a file
SMBlock	lock a range of bytes in a file
SMBunlock	unlock a range of bytes in a file
SMBflush	force any buffers of a file to disk
SMBclose	close a file
SMBmv	rename a file
SMBunlink	delete a file

7.1 SMBcreate Specification

SMBcreate Detailed Description

This core protocol request is used to create and open a new regular file, or open an existing regular file and truncate its length to zero. The file-sharing mode for the open operation cannot be specified. The FID returned can be used in subsequent commands.

SMBcreate Deviations

1. The archive, system and hidden file attribute bits may be ignored, in accordance with the File Attribute mapping convention (see Section 4.3.1 on page 30).
2. The create time specified is used to set the LMX server's last modify time for the file.

SMBcreate Field Descriptions

From SMB redirector		To SMB redirector	
Field Name	Field Value	Field Name	Field Value
smb_com	*SMBcreate*	*smb_com*	*SMBcreate*
smb_wct	3	*smb_wct*	1
smb_vwv[0]	*smb_attr*	*smb_vwv*[0]	*smb_fid*
smb_vwv[1-2]	*smb_time*	*smb_bcc*	0
smb_bcc	min=2		
smb_buf[ ]	*smb_pathname*		

smb_attr This is a file attribute field (see Section 5.3.3 on page 43). It defines the attributes to be given to the newly-created file. The bits 3 and 4 (volume label and directory) are not allowed to be set. If the file already exists, this field is ignored.

smb_time A 32-bit integer which sets the LMX server's idea of the last modify time for the file. A value of zero indicates a null time field (see Section 5.3.1 on page 43).

smb_pathname An ASCIIZ (type 04) buffer containing the name of the file to be created.

smb_fid This signed integer is the FID returned by the LMX server for the opened file. The SMB redirector will use that FID in other requests to refer to this particular file.

SMBcreate Error Code Descriptions

CAE Code	DOS Class	DOS Code	Description
EACCES	ERRDOS	ERRnoaccess	File does not exist and the directory in which the file is to be created does not permit writing.
EACCES	ERRDOS	ERRnoaccess	Search permission is denied on a component of the path-prefix.
EACCES	ERRDOS	ERRnoaccess	File exists and write permission is denied.
EAGAIN	ERRDOS	ERRbadshare	File exists, mandatory file/record locking is set, and there are outstanding record locks on the file.
EFAULT	ERRSRV	ERRerror	Path points outside the allocated address space of the process.
EINTR	ERRSRV	ERRerror	A signal was caught during the operation.
EISDIR	ERRDOS	ERRnoaccess	Named file is an existing directory.
EMFILE	ERRDOS	ERRnofids	Maximum number of file descriptors are currently open in this process.
ENFILE	ERRDOS	ERRnofids	System file table is full.
ENOENT	ERRDOS	ERRbadfile	Component of path-prefix does not exist or pathname is null.
ENOSPC	ERRSRV	ERRerror	File must be created, and the system is out of resources necessary to create files.
ENOTDIR	ERRDOS	ERRbadpath	Component of path-prefix is not a directory.
ENXIO	ERRSRV	ERRerror	Named file is a character-special or block-special file and the device associated with this special file does not exist; or O_NDELAY is set, file is a FIFO, O_WRONLY is set and no process has the file open for reading.
EROFS	ERRSRV	ERRerror	Named file resides on read-only file system.
ETXTBSY	ERRSRV	ERRaccess	File is a pure procedure file that is being executed.
-	ERRSRV	ERRinvtid	TID specified in command is invalid.
-	ERRSRV	ERRinvdevice	File creation request made to a share that is not a file system subtree.
-	ERRSRV	ERRaccess	Named file exists as a directory, special file or named pipe.
-	ERRSRV	ERRaccess	Write and Create permissions required, or the file attributes specified a volume label.
-	ERRSRV	ERRbaduid	The UID given (*smb_uid*) is not known as a valid ID on this LMX session.
-	SUCCESS	SUCCESS	Everything worked, no problems.

SMBcreate Preconditions

1. The SMB redirector has sent a valid SMB request with a valid TID for a file system subtree and valid UID.
2. The SMB redirector must have write permission on the file's parent directory in order to create a new file, or write permission on the file itself in order to truncate it. The permission is granted via the security mode used (refer to Section 3.3 on page 12).

SMBcreate Postconditions

1. The LMX server obeys the rules for mapping the new file into the CAE file system. If the read-only attribute is set, the CAE write permission bits for the mode of the file are turned off.
2. The LMX server's last modify time for the file will be set according to *smb_time*. If *smb_time* was zero, the last modify time for the file will be left unchanged.
3. The SMB redirector will be granted read/write access to the file if it was created (even if the read-only bit was set). If the file existed, access rights will be granted according to the existing access mode.
4. The newly-created or truncated file is opened in the DOS read/write compatibility mode.

SMBcreate Side Effects

File is created or truncated.

Conventions

- Attribute (see Section 4.3.1 on page 30).
- Filename (see Section 3.5 on page 15).
- Opportunistic Locking (see Section 3.8.2 on page 20).

7.2 SMBmknew Specification

SMBmkoew Detailed Description

This core protocol request is equivalent to the *SMBcreate* request except that it will fail if the named file already exists.

SMBmkoew Deviations

1. The archive, system and hidden file attribute bits are ignored.
2. The create time specified is used to set the LMX server's last modify time for the file.

SMBmkoew Field Descriptions

From SMB redirector		To SMB redirector	
Field Name	Field Value	Field Name	Field Value
smb_com	*SMBmknew*	*smb_com*	*SMBmknew*
smb_wct	3	*smb_wct*	1
smb_vwv[0]	*smb_attr*	*smb_vwv*[0]	*smb_fid*
smb_vwv[1-2]	*smb_time*	*smb_bcc*	0
smb_bcc	min=2		
smb_buf[]	*smb_path*		

smb_attr A file attribute field (refer to Section 5.3.3 on page 43) containing attributes to be given to the new file. The bits 3 and 4 (volume label and directory) are not allowed to be set.

smb_time A 32-bit integer to be used as the file creation time.

smb_path An ASCIIZ (type 04) buffer containing the name of the file to be created.

smb_fid A 16-bit integer containing the FID the SMB redirector will use to refer to the opened file.

SMBmknew Error Code Descriptions

CAE Code	DOS Class	DOS Code	Description
EACCES	ERRDOS	ERRnoaccess	Search permission is denied on a component of the path-prefix, or the parent directory does not permit writing.
EACCES	ERRDOS	ERRnoaccess	Requested permission is denied for the named file.
EEXIST	ERRDOS	ERRnoaccess	O_CREAT and O_EXCL are set and the file exists.
EFAULT	ERRSRV	ERRerror	Path points outside the allocated address space of the process.
EINTR	ERRSRV	ERRerror	A signal was caught during the operation.
EMFILE	ERRDOS	ERRnofids	Maximum number of file descriptors are currently open in this process.
ENFILE	ERRDOS	ERRnofids	System file table is full.
ENOENT	ERRDOS	ERRbadfile	Component of path-prefix does not exist.
ENOSPC	ERRSRV	ERRerror	The system is out of resources necessary to create files.
ENOTDIR	ERRDOS	ERRbadpath	Component of path-prefix is not a directory.
EROFS	ERRSRV	ERRerror	Named file resides on read-only file system.
-	ERRSRV	ERRaccess	Write and create permissions for the directory required.
-	ERRSRV	ERRinvtid	TID specified in command is invalid.
-	ERRSRV	ERRinvdevice	File creation request made to a share that is not a file system subtree.
-	ERRSRV	ERRbaduid	The UID given (*smb_uid*) is not known as a valid ID on this LMX session.
-	SUCCESS	SUCCESS	Everything worked, no problems.

SMBmknew Preconditions

1. The SMB redirector has sent a valid SMB request, with a valid UID and valid TID for a file system subtree.
2. The SMB redirector must have appropriate permissions in order to create the new file.
3. The named file must not exist before the request is sent.

SMBmknew Postconditions

1. A new file with the given pathname will be created and opened, or an error will be returned.
2. The LMX server obeys the rules for mapping the new file into the CAE file system. If the read-only file attribute is set, the CAE write permission bit of the mode for the new file must be turned off.
3. The LMX server's last modify time for the file will be set to *smb_time*. If *smb_time* is zero, the LMX server will assign the current time.
4. The SMB redirector is granted read/write access to the file regardless of *smb_attr*.
5. The newly-created file is opened in DOS read/write compatibility mode.

SMBmkoew Side Effects

None.

Coovention s

- Attribute (see Section 4.3.1on page 30).
- Filename (see Section 3.5on page 15).
- Opportunistic Locking (see Section 3.8.2on page 20).

7.3 SMBopen Specification

SMBopen Detailed Description

This core protocol request is used to open an existing regular file and obtain an FID which is used to refer to the file in subsequent requests. It cannot be used to open directories or LMX named pipes (refer to the X/Open CAE Specification, IPC Mechanisms for SMB).

SMBopen Deviations

The archive, system and hidden file attribute bits in the output attribute field are treated according to Section 4.3.1 on page 30.

SMBopen Field Descriptions

From SMB redirector		To SMB redirector	
Field Name	Field Value	Field Name	Field Value
smb_com	*SMBopen*	*smb_com*	*SMBopen*
smb_wct	2	*smb_wct*	7
smb_vwv[0]	*smb_mode*	*smb_vwv*[0]	*smb_fid*
smb_vwv[1]	*smb_iattr*	*smb_vwv*[1]	*smb_oattr*
smb_bcc	min=2	*smb_vwv*[2,3]	*smb_time*
smb_buf[]	*smb_path*	*smb_vwv*[4,5]	*smb_size*
		smb_vwv[6]	*smb_access*
		smb_bcc	0

smb_mode A file-sharing control field which indicates the access modes and deny modes being requested (see Section 5.3.5 on page 44).

smb_iattr Attributes to be assigned to the file. Ignored.

smb_path An ASCIIZ (type 04) buffer containing the name of the file to be opened.

smb_fid A 16-bit signed integer containing the FID returned for the opened file.

smb_oattr Attributes currently assigned to the file (see Section 5.3.3 on page 43).

smb_time A 32-bit integer time of the last modification to the opened file (see Section 5.3.1 on page 43).

smb_size A 32-bit signed integer which contains the current size of the opened file, in bytes.

smb_access An access mode field (see Section 5.3.7 on page 46) indicating the access permission set actually granted to the opening process.

SMBopen Error Code Descriptions

CAE Code	DOS Class	DOS Code	Description
EACCES	ERRDOS	ERRnoaccess	Search permission is denied on a component of the path-prefix.
EACCES	ERRDOS	ERRnoaccess	Requested access permission is denied for the named file.
EAGAIN	ERRDOS	ERRbadshare	File exists, mandatory file/record locking is set, and there are outstanding record locks on the file.
EFAULT	ERRSRV	ERRerror	Path points outside the allocated address space of the process.
EINTR	ERRSRV	ERRerror	A signal was caught during the open operation.
EISDIR	ERRDOS	ERRnoaccess	Named file is a directory and oflag is write or read/write.
EMFILE	ERRDOS	ERRnofids	Maximum number of file descriptors are currently open in this process.
ENFILE	ERRDOS	ERRnofids	System file table is full.
ENOENT	ERRDOS	ERRbadfile	File does not exist, or component of pathname does not exist.
ENOTDIR	ERRDOS	ERRbadpath	Component of path-prefix is not a directory.
ENXIO	ERRSRV	ERRerror	Generic LMX server open failure.
EROFS	ERRSRV	ERRerror	Named file resides on read-only file system and requested access permission is write or read/write.
ETXTBSY	ERRDOS	ERRnoaccess	File is a pure procedure file that is being executed and requested access permission specifies write or read/write.
-	ERRSRV	ERRaccess	Permission conflict between requested permission and permissions for the shared resource; for example, open for write of a file in a read-only file system subtree.
-	ERRSRV	ERRinvnid	TID specified in command is invalid.
-	ERRSRV	ERRinvdevice	File creation request made to a share that is not a file system subtree.
-	ERRSRV	ERRbaduid	The UID given (*smb_uid*) is not known as a valid ID on this LMX session.
-	ERRDOS	ERRnoaccess	Open mode failure. See rules for Compatibility and DENY mode opens.
-	SUCCESS	SUCCESS	Everything worked, no problems.

SMBopen Preconditions

1. The SMB redirector has sent a valid SMB request, with a valid UID and a valid TID.
2. The file being opened must exist.
3. The pathname specified is not an LMX named pipe.

SMBopen Postconditions

1. The file will be opened in the requested mode with the returned FID, or an error will be returned.
2. The file will be opened only if the user has the appropriate permissions and there is no conflict between already-granted access or deny modes and the requested access or deny modes.

SMBopen Side Effects

The file exclusion mode requested will be in effect for subsequent open commands.

Conventions

- Access (see Section 4.3.2 on page 30).
- Attribute (see Section 4.3.1 on page 30).
- Filename (see Section 3.5 on page 15).
- Opportunistic Locking (see Section 3.8.2 on page 20).

7.4 SMBread Specification

SMBread Detailed Description

This core protocol request will read bytes from a regular file and, if an extended protocol is negotiated, from a named pipe, mailslot or directly accessible device. End-of-file is indicated by returning fewer bytes than requested; a read starting at or beyond end-of-file returns zero bytes.

SMBread Deviations

None.

SMBread Field Descriptions

From SMB redirector		To SMB redirector	
Field Name	Field Value	Field Name	Field Value
smb_com	*SMBread*	*smb_com*	*SMBread*
smb_wct	5	*smb_wct*	5
smb_vwv[0]	*smb_fid*	*smb_vwv*[0]	*smb_count*
smb_vwv[1]	*smb_bytecount*	*smb_vwv*[1-4]	rsvd (MBZ)
smb_vwv[2-3]	*smb_offset*	*smb_bcc*	length of data + 3
smb_vwv[4]	*smb_countleft*	*smb_buf*[]	*smb_data*
smb_bcc	0		

smb_fid A 16-bit signed integer indicating the file from which *smb_data* should be read.

smb_bytecount A 16-bit unsigned integer indicating the amount of data to be read. The SMB redirector will ensure that the amount requested will fit in the negotiated maximum buffer size.

smb_offset A 32-bit unsigned integer defining the file pointer position.

smb_countleft A 16-bit unsigned integer. This field is advisory, and some SMB redirectors will set it to zero, in which case it should be ignored. If the value is not zero, then it is an estimate of the total number of bytes that will be read, including those read by this request. This additional information may be used by the LMX server to optimise buffer allocation and/or read-ahead.

smb_count A 16-bit unsigned integer giving the actual number of bytes returned to the SMB redirector. This must be equal to *smb_bytecount*, unless:

1. End-of-file was reached before reading *smb_bytecount* bytes. The number of bytes actually read, along with that data, is returned.
2. *smb_offset* pointed at or beyond end-of-file. A zero (0) value is returned.

rsvd These four 16-bit fields are reserved and must be zero.

smb_data A Data Block (type 01) buffer containing the actual data read from the file (see Section 5.3.4 on page 44).

SMBread Error Code Descriptions

CAE Code	DOS Class	DOS Code	Description
EIO	ERRHRD	ERRdata	A problem has occurred in the physical I/O.
ENXIO	ERRHRD	ERRwrite	The device associated with the file descriptor is a block-special or character-special file and the value of the file pointer is out of range.
EBADF	ERRSRV	ERRerror	An FID was validated by the LMX server but unacceptable to the system.
EAGAIN	ERRDOS	ERRlock	O_NDELAY set and (a) read from empty CAE FIFO attempted, or (b) file open on the LMX server and a record lock on the file exists.
EDEADLK	ERRSRV	ERRerror	The read would block and deadlock would result.
ENOLCK	ERRDOS	ERRnoaccess	File is open on the LMX server in enforced-lock mode, a record lock exists on the file, and the file was opened with O_NDELAY set.
-	ERRDOS	ERRnoaccess	Attempt to read from a portion of the file that the LMX server knows has been locked or been opened in deny-read.
-	ERRDOS	ERRbadaccess	Read permission required.
-	ERRDOS	ERRbadfid	Attempt to read from an FID that the LMX server does not have open.
-	ERRSRV	ERRerror	Corrupt SMB request has been encountered.
-	ERRSRV	ERRinvdevice	Attempt to read from an open spool file.
-	ERRSRV	ERRinvnid	Invalid TID in request.
-	ERRSRV	ERRbaduid	The UID given (*smb_uid*) is not known as a valid ID on this LMX session.
-	SUCCESS	SUCCESS	Everything worked, no problems.

SMBread Preconditions

1. The SMB redirector has sent a valid SMB request.
2. The SMB redirector's read request will fit in an SMB buffer of the negotiated size.
3. The SMB redirector must have a valid TID for a file system resource with the appropriate permissions for the read operation.
4. The SMB redirector must have a valid FID and at least read access.

SMBread Postconditions

1. If the read was successful, the LMX server has returned to the SMB redirector either the data for all of the requested read or all the data that was available up to the EOF.
2. If the read failed, the LMX server has returned to the SMB redirector an SMB response indicating the reason for the failure of this read or a previous block operation.

SMBread Side Effects

None.

Conventions

- Locking (see Section 4.4 on page 33).

7.5 SMBwrite Specification

SMBwrite Detailed Description

This core protocol request writes bytes from a regular file and, if an extended protocol is negotiated, to a named pipe, mailslot or directly accessible device. It can also be used to truncate a file to a given point or extend a file beyond its current size.

SMBwrite Deviations

None.

SMBwrite Field Descriptions

From SMB redirector		To SMB redirector	
Field Name	Field Value	Field Name	Field Value
smb_com	*SMBwrite*	*smb_com*	*SMBwrite*
smb_wct	5	*smb_wct*	1
smb_vwv[0]	*smb_fid*	*smb_vwv*[0]	*smb_count*
smb_vwv[1]	*smb_bytecount*	*smb_bcc*	0
smb_vwv[2,3]	*smb_offset*		
smb_vwv[4]	*smb_countleft*		
smb_bcc	length of data + 3		
smb_buf[]	*smb_data*		

smb_fid The FID to be written to.

smb_bytecount An unsigned integer indicating the number of bytes to be written. If this value is zero, the file should be truncated or extended to the size indicated in *smb_offset*. If extended, the bytes between the old and new EOF will be zero.

smb_offset A 32-bit unsigned integer defining the file position at which the data should be written.

smb_countleft A 16-bit unsigned integer. This field is advisory, and some SMB redirectors will set it to zero, in which case it should be ignored. If the value is not zero, then it is an estimate of the total number of bytes that will be written, including those written by this request. This additional information may be used by the LMX server to optimise buffer allocation or perform write-behind.

smb_data A Data Block (type 01) buffer containing the actual bytes to be written (see Section 5.3.4 on page 44).

smb_count A 16-bit unsigned integer containing the actual number of bytes written. If this is less than *smb_bytecount* but no explicit error is returned, then insufficient file system space prevented more than *smb_count* of bytes from being written.

SMBwrite Error Codes

CAE Code	DOS Class	DOS Code	Description
EIO	ERRHRD	ERRdata	A problem occurred during physical I/O.
ENXIO	ERRHRD	ERRwrite	An error occurred on the FID being written to.
EBADF	ERRDOS	ERRbadfid	A valid *smb_fid* mapped to an LMX server FID not accepted by the operating system.
EAGAIN	ERRDOS	ERRnoaccess	Resources for I/O temporarily exhausted
EFBIG	SUCCESS	SUCCESS	The file has grown too large (size exceeds *ulimit*) and no more data can be written to the file. An *smb_count* of 0 will be returned to the SMB redirector in the count field of the SMB response. This indicates to the SMB redirectors that the file system is full.
ENOSPC	SUCCESS	SUCCESS	No space on the file system; *smb_count* will be 0, indicating the file system is full.
EPIPE	ERRHRD	ERRbadunit	Write to a named pipe with no reader.
EDEADLK	ERRSRV	ERRerror	The write would block due to locking, but O_NDELAY was set.
ERANGE	ERRSRV	ERRerror	Attempted write size is outside of the minimum and maximum ranges that can be written to the supplied FID.
ENOLCK	ERRDOS	ERRnoaccess	A record lock has been taken on the file, or the SMB redirector has attempted to write to a portion of the file that the LMX server knows has been locked, opened in deny-write open mode, or opened in read-only mode.
-	ERRDOS	ERRbadaccess	Write permission required.
-	ERRDOS	ERRbadfid	Invalid FID specified.
-	ERRSRV	ERRerror	Corrupt SMB request was received.
-	ERRSRV	ERRinvdevice	Attempt to write to an open spool file.
-	ERRSRV	ERRinvtid	Invalid TID specified.
-	ERRSRV	ERRbaduid	The UID given (*smb_uid*) is not known as a valid ID on this LMX session.
-	SUCCESS	SUCCESS	Everything worked, no problems.

SMBwrite Preconditions

1. The SMB redirector has sent a valid SMB request.
2. The SMB redirector's write request will fit in an SMB buffer.
3. The SMB redirector must have a valid TID to a regular file system resource with appropriate permissions for the write operation.
4. The SMB redirector must have a valid FID with at least write access.

SMBwrite Postconditions

1. If the write was successful, the LMX server has returned to the SMB redirector either a count value for a write of the entire amount or a count value for less than the entire write amount if file system space is exhausted or the file has reached the maximum file size.
2. If the write failed, the LMX server has returned to the SMB redirector an SMB request indicating the reason for the failure of this write or a previous block operation.

SMBwrite Side Effects

The data is not necessarily reflected in the file system until an *SMBflush* or the FID is closed.

Conventions

- Locking (see Section 4.4 on page 33).

7.6 SMBlseek Specification

SMBlseek Detailed Description

The *SMBlseek* core protocol request sets the current file pointer for a regular file. The response returns the new file pointer expressed as the offset from the start of the file, and may be beyond the current end-of-file. An attempt to seek to a position before the beginning-of-file sets the file pointer to beginning-of-file.

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

SMBlseek Deviations

None.

SMBlseek Field Descriptions

From SMB redirector		To SMB redirector	
Field Name	Field Value	Field Name	Field Value
smb_com	*SMBlseek*	*smb_com*	*SMBlseek*
smb_wct	4	*smb_wct*	2
smb_vwv[0]	*smb_fid*	*smb_vwv*[0-1]	*smb_offset*
smb_vwv[1]	*smb_mode*	*smb_bcc*	0
smb_vwv[2-3]	*smb_offset*		
smb_bcc	0		

smb_fid The FID whose pointer is to be manipulated.

smb_mode A 16-bit field indicating where (beginning=0, current position=1, end=2) the seek is to take place.

smb_offset A 32-bit signed integer. In the request, indicates how far to move from the position indicated by *smb_mode*. Positive values move forward in the file towards EOF; negative values move backward through the file towards BOF. In the response, indicates the resulting position after the move, relative to BOF.

SMBlseek Error Code Descriptions

CAE Code	DOS Class	DOS Code	Description
EBADF	ERRDOS	ERRbadfid	FID is valid but not accepted by the system.
EINVAL	ERRDOS	ERRnoaccess	Invalid *smb_mode*.
ESPIPE	ERRDOS	ERRnoaccess	Cannot seek on this file (named pipe).
-	ERRDOS	ERRbadfid	The SMB redirector has supplied an invalid FID.
-	ERRDOS	ERRnoaccess	The SMB redirector's context does not permit this access.
-	ERRSRV	ERRinvnid	TID specified in command is invalid.
-	ERRSRV	ERRinvdevice	Attempt to seek on a non-regular file.
-	ERRSRV	ERRerror	The LMX server has received a corrupt SMB request.
-	ERRSRV	ERRbaduid	The UID given (*smb_uid*) is not known as a valid ID on this LMX session.
-	SUCCESS	SUCCESS	Everything worked, no problems.

SMBlseek Preconditions

1. The SMB redirector has sent a valid SMB request with a valid TID for a file system resource.
2. The SMB redirector must have acquired a valid FID from the LMX server.
3. The SMB redirector has specified a valid *smb_mode* value.

SMBlseek Postconditions

1. If the *SMBlseek* was successful, the LMX server has returned to the SMB redirector the new file pointer position.
2. If the *SMBlseek* was unsuccessful, the LMX server has returned an error indicating the failure of this operation or of a previous block operation.

SMBlseek Side Effects

The current file position maintained by the LMX server is changed to the offset returned to the SMB redirector.

Conventions

None.

7.7 SMBlock Specification

SMBlock Detailed Description

This command is sent by an SMB redirector process to lock a given byte range of a regular file. A lock prevents attempts to lock, read or write the byte range by any other SMB redirector. Multiple non-overlapping lock ranges are allowed on the same file. Overlapping locks are not allowed. Byte ranges beyond the current end-of-file may be locked; however, such locks will not cause allocation of file space. A lock may only be unlocked by the process (PID) that performed the lock.

SMBlock Deviations

Refer to Section 4.4 on page 33

SMBlock Field Descriptions

From SMB redirector		To SMB redirector	
Field Name	Field Value	Field Name	Field Value
smb_com	*SMBlock*	*smb_com*	*SMBlock*
smb_wct	5	*smb_wct*	0
smb_vwv[0]	*smb_fid*	*smb_bcc*	0
smb_vwv[1-2]	*smb_count*		
smb_vwv[3 4]	*smb_offset*		
smb_bcc	0		

smb_fid The FID to be locked.

smb_count A 32-bit unsigned integer containing the number of bytes in the lock range.

smb_offset A 32-bit unsigned integer containing the offset to the start of the lock range.

SMBlock Error Code Descriptions

CAE Code	DOS Class	DOS Code	Description
EBADF	ERRSRV	ERRerror	A valid FID was rejected by the underlying system.
EACCES	ERRDOS	ERRnoaccess	File access rights do not match requested locks.
EACCES	ERRDOS	ERRlock	A lock has already been taken out on this record.
ENOLCK	ERRDOS	ERRlock	Insufficient resources to place the requested lock.
EDEADLK	ERRSRV	ERRerror	The lock request would block and cause a deadlock with another process.
-	ERRDOS	ERRbadfid	An invalid FID was specified.
-	ERRDOS	ERRlock	Byte range is already locked by another serving process.
-	ERRSRV	ERRerror	An invalid SMB request was sent.
-	ERRSRV	ERRinvnid	TID specified in command is invalid.
-	ERRSRV	ERRinvdevice	Attempt to lock on a non-regular file.
-	ERRSRV	ERRbaduid	The UID given (*smb_uid*) is not known as a valid ID on this LMX session.
-	SUCCESS	SUCCESS	Everything worked, no problems.

SMBlock Preconditions

1. The SMB redirector has sent a valid SMB request with valid access to the file system subtree.
2. The SMB redirector must have a valid FID.

SMBlock Postconditions

The given byte range of the file will be locked preventing access by other SMB redirectors not using the same FID.

SMBlock Side Effects

Only requests using the PID as sent in the *SMBlock* request may access the locked record(s).

Conventions

- Locking (see Section 4.4 on page 33).

7.8 SMBunlock Specification

SMBunlock Detailed Description

This core protocol request is used to unlock a byte range. The byte range specified must be exactly the same as that specified in a previous successful lock request from the same SMB redirector process (that is, the PID must be the same). An unlock request for a range that was not locked is treated as an error.

SMBunlock Deviations

None.

SMBunlock Field Descriptions

From SMB redirector		To SMB redirector	
Field Name	Field Value	Field Name	Field Value
smb_com	*SMBunlock*	*smb_com*	*SMBunlock*
smb_wct	5	*smb_wct*	0
smb_vwv[0]	*smb_fid*	*smb_bcc*	0
smb_vwv[1-2]	*smb_count*		
smb_vwv[3-4]	*smb_offset*		
smb_bcc	0		

This request is identical in format to *SMBlock* (see Section 7.7 on page 81).

SMBunlock Error Code Descriptions

Additional applicable error codes can be found in the specification of *SMBlock* (see Section 7.7 on page 81).

CAE Code	DOS Class	DOS Code	Description
-	ERRDOS	ERRlock	The record cannot be unlocked with this PID or a lock on this range does not exist for this PID.
-	SUCCESS	SUCCESS	Everything worked, no problems.

SMBunlock Preconditions

1. The SMB redirector has sent a valid SMB request with a valid TID for a file system resource.
2. The SMB redirector must have a valid FID.
3. The byte range and PID specified must exactly match a byte range and PID specified in a previous successful lock operation on this FID.

SMBunlock Postconditions

The specified byte range of the file will be unlocked, or an error will be returned.

SMBunlock Side Effects

The record is now open for reading/writing/locking by other SMB redirectors.

Conventions

- Locking (see Section 4.4 on page 33).

7.9 SMBflush Specification

SMBflush Detailed Description

This core request flushes data and allocation information for a specified file or for all files open under this LMX session.

SMBflush Deviations

Some CAE systems provide no way for a programme to block until the local file cache has actually flushed to the disk, but simply indicate that a flush has been scheduled and will complete soon. An LMX server should nonetheless take steps to maximise the probability that the data is truly on disk before the SMB redirector is notified.

An LMX server may always flush all files supported on the LMX session even if a single-file flush was requested.

SMBflush Field Descriptions

From SMB redirector		To SMB redirector	
Field Name	Field Value	Field Name	Field Value
smb_com	*SMBflush*	*smb_com*	*SMBflush*
smb_wct	1	*smb_wct*	0
smb_vwv[0]	*smb_fid*	*smb_bcc*	0
smb_bcc	0		

smb_fid The FID to be flushed. If this field is set to 0xffff (that is, – 1), all files open in the LMX session environment will be flushed.

SMBflush Error Code Descriptions

CAE Code	DOS Class	DOS Code	Description
-	ERRSRV	ERRinvnid	Bad TID.
-	ERRDOS	ERRbadfid	The specified FID is not open.
-	ERRSRV	ERRerror	Other CAE errors mapped here.
-	ERRSRV	ERRbaduid	The UID given (*smb_uid*) is not known as a valid ID on this LMX session.
-	SUCCESS	SUCCESS	Everything worked, no problems.

SMBflush Preconditions

1. The SMB redirector must have issued a valid SMB request with a valid UID and valid TID for a shared resource.
2. The specified FID must be open, or it must be 0xffff.

SMBflush Postconditions

1. All modified data and retrieval state information is scheduled to be flushed to stable store.
2. Buffered named pipe data, if any, is flushed through to the cooperating processes.

SMBflush Side Effects

Eventually, the data will be written to stable store.

Conventions

None.

7.10 SMBclose Specification

SMBclose Detailed Description

This core protocol request is sent by an SMB redirector process to invalidate the given FID for that process. All locks held by the SMB redirector process on that FID will be released as part of the close. The FID cannot be used by the SMB redirector for further file access requests.

SMBclose Deviations

None.

SMBclose Field Descriptions

From SMB redirector		To SMB redirector	
Field Name	Field Value	Field Name	Field Value
smb_com	*SMBclose*	*smb_com*	*SMBclose*
smb_wct	3	*smb_wct*	0
smb_vwv[0]	*smb_fid*	*smb_bcc*	0
smb_vwv[1-2]	*smb_time*		
smb_bcc	0		

smb_fid The FID to be closed.

smb_time An LMX server may optionally update the last modification time for the file to *smb_time*. A zero (0) or 0xffffffff *smb_time* results in the LMX server using the default value.

SMBclose Error Code Descriptions

CAE Code	DOS Class	DOS Code	Description
EBADF	ERRDOS	ERRbadfid	The FID is valid but no longer accepted by the operating system.
-	ERRDOS	ERRbadfid	The SMB redirector has supplied an invalid FID.
-	ERRSRV	ERRinvnid	TID specified in command is invalid.
-	ERRSRV	ERRinvdevice	Attempt to close an open spool file.
-	ERRSRV	ERRbaduid	The UID given (*smb_uid*) is not known as a valid ID on this LMX session.
-	SUCCESS	SUCCESS	Everything worked, no problems.

SMBclose Preconditions

1. The SMB redirector has sent a valid SMB request, with a valid UID and TID.
2. The SMB redirector has sent a valid FID for an open file.

SMBclose Postconditions

1. If the file being closed was written to, all the modified buffers for the file will be flushed to the file system.
2. Any remaining locks on the FID (including opportunistic locks) will be removed.
3. The last modify time for the file will be set to the time specified by the SMB redirector.
4. The FID will be invalidated for further file access requests.

SMBclose Side Effects

None.

Conventions

None.

7.11 SMBmv Specification

SMBmv Detailed Description

This core protocol request changes the name of one or more files or directories. Multiple files may be renamed in response to a single request, as *SMBmv* supports filenames with wildcards in the last 8.3component of the pathname; wildcards elsewhere in pathnames are not permitted.

Every file that matches the attribute field and the first pathname is renamed according to the second pathname, provided that file does not already exist (see Section 3.6on page 17 for more details of the name transformation).

Wildcards are not allowed in the destination path for directories. A move of a directory cannot have a destination located in the directory itself or any subdirectory within the source directory. In these conditions the error <ERRDOS, ERRbadpath> is to be returned.

If a * is received it indicates to the LMX server to fill the remainder of the component with ?. Any characters provided after the * will be ignored and the usual ? wildcard mapping applies.

A file to be renamed can be open. If it is opened by the requesting process, the open must be in compatibility mode. Otherwise, the rename fails with <ERRDOS, ERRnoaccess>. If the file is opened by another process, that process has an oplock on the file, and the process has asked for extended notification, the rename request will block until after the oplock has been broken. If the process with the oplock closed the file, the rename takes place; if not, it fails.

There must not already be a different file existing with the new name. If there is, the rename will fail. If wildcards are used in a rename operation, and only some of the renames fail for any reason, the request will fail silently; that is, no error will be returned.

Because an LMX server may serve multiple requests on the same resource simultaneously, there may be interactions between the execution of this request and ongoing searches of the same resource (*SMBsearch*, *SMBffirst*, *SMBfunique*, *SMBfclose*). Although there is no prohibition on renaming directories actively being searched, an LMX server may cause the search to appear to have reached the end of the directory since no more entries will be found.

SMBmv Deviations

Some LMX servers will ignore the attribute field; others treat it according to the Attribute convention.

An LMX server may choose to return the error <ERRDOS, ERRdiffdevice> if the move requested spans two different CAE file systems.

SMBmv Field Descriptions

From SMB redirector		To SMB redirector	
Field Name	Field Value	Field Name	Field Value
smb_com	*SMBmv*	*smb_com*	*SMBmv*
smb_wct	1	*smb_wct*	0
smb_vwv[0]	*smb_attr*	*smb_bcc*	0
smb_bcc	min = 4		
smb_buf[]	*smb_oldpath*		
	smb_newpath		

smb_attr A file attribute field. An LMX server should match file attributes against this field when selecting files which match *smb_oldpath* to rename. Items that match this field are added with regular files to the list of items moved.

smb_oldpath An ASCIIZ (type 04) buffer containing the name of the file or files to be renamed. Only the filename component (not directory components) may contain wildcards.

smb_newpath An ASCIIZ (type 04) buffer containing the new name(s) to be given to the file(s) which match *smb_oldpath*.

SMBmv Error Code Descriptions

CAE Code	DOS Class	DOS Code	Description
ENOTDIR	ERRDOS	ERRbadpath	A component in the old pathname is not a directory.
ENOENT	ERRDOS	ERRbadfile	The old file does not exist.
EACCES	ERRSRV	ERRaccess	A component in a pathname denies the required permission.
EEXIST	ERRDOS	ERRnoaccess	The new file already exists.
EXDEV	ERRDOS	ERRdiffdevice	Attempt to rename to a different device.
EROFS	ERRHRD	ERRnowrite	Attempt to write on a read-only file system.
EMLINK	ERRDOS	ERRnoaccess	Too many links to old file.
ENOSPC	ERRDOS	ERRnoaccess	The directory is full.
EBUSY	ERRDOS	ERRnoaccess	The old path is the mounted point for a file system.
ETXTBSY	ERRDOS	ERRnoaccess	The old path is the last link to an executing programme.
-	ERRSRV	ERRaccess	An attempt was made to change a volume label.
	ERRSRV	ERRerror	Internal error.
-	ERRSRV	ERRbaduid	The UID given (*smb_uid*) is not known as a valid ID on this LMX session.
-	SUCCESS	SUCCESS	Everything worked, no problems.

SMBmv Preconditions

1. SMB, UID and TID are valid; TID is for a file system resource.
2. *smb_oldpath* must refer to one or more files.
3. Transformation with *smb_newpath* must not match any existing files.
4. Process has appropriate permissions for all directories in both path arguments; write permissions on last directory in each path argument.

SMBmv Postconditions

smb_oldpath no longer points to any existing files. (This condition may not persist in the presence of other file-sharing activity, or if some of the new names conflicted with already-existing files.)

SMBmv Side Effects

Searches involving renamed directories may be prematurely terminated.

Conventions

- Access (see Section 4.3.2 on page 30).
- Attribute (see Section 4.3.1 on page 30).
- Filename (see Section 3.5 on page 15).
- Opportunistic Locking (see Section 3.8.2 on page 20).
- Wildcards (see Section 3.6 on page 17).

7.12 SMBunlink Specification

SMBunlink Detailed Description

This core protocol request is sent to delete a regular file or files. Read-only files may not be deleted unless the read-only attribute is set in the *SMBunlink* request. Wildcards in the filename part of the pathname are supported.

The effect of the *SMBunlink* will be LMX server implementation-dependent. Normally only the referenced filename can be deleted. If another SMB redirector has the file open, the contents of the file will remain available until that SMB redirector closes the handle to the file. If opportunistic locking is supported and another SMB redirector has been granted an oplock on the file, the process has asked for notification of the *SMBunlink* request. The *SMBunlink* request being processed will block until the oplock has been broken (reference Section 3.8.2 on page 20).

If a wildcard pathname matches more than one file, and not all of the files could be unlinked, the request fails silently.

The *smb_attr* field may be applied as an additional filter on files matching the wildcard string in *smb_path*. LMX servers may optionally provide this filtering function.

SMBunlink Deviations

Only the specified directory entry is immediately deleted. The file contents are deleted only when all the file's directory entries have been deleted and all the FIDs associated with it have been destroyed.

Some LMX servers may ignore the *smb_attr* field. Others will treat it in accordance with the attribute convention (refer to Section 3.7 on page 17).

LMX servers require the user to have write permission in the target file's parent directory.

SMBunlink Field Descriptions

From SMB redirector		To SMB redirector	
Field Name	Field Value	Field Name	Field Value
smb_com	*SMBunlink*	*smb_com*	*SMBunlink*
smb_wct	1	*smb_wct*	0
smb_vwv[0]	*smb_attr*	*smb_bcc*	0
smb_bcc	min = 2		
smb_buf[]	*smb_path*		

smb_attr A file attribute field. Some LMX servers treat it as indicating the attributes that the target file must have.

smb_path An ASCIIZ (type 04) buffer indicating the file to be unlinked.

SMBunlink Error Code Descriptions

CAE Code	DOS Class	DOS Code	Description
ENOTDIR	ERRDOS	ERRbadpath	A component in the path-prefix is not a directory.
ENOENT	ERRDOS	ERRbadfile	The specified file does not exist.
EACCES	ERRSRV	ERRaccess	A component in the path denies the required permission.
EPERM	ERRDOS	ERRnoaccess	The specified file is a directory.
EROFS	ERRHRD	ERRnowrite	Attempt to modify a read-only file system.
EBUSY	ERRDOS	ERRnoaccess	The specified file is a directory.
ETXTBUSY	ERRDOS	ERRnoaccess	The specified file is the last link to a shared text file.
-	ERRSRV	ERRaccess	Attempt to delete a volume label, or delete permission required.
-	ERRSRV	ERRinvdevice	Attempt to unlink a non-regular file.
	ERRSRV	ERRerror	Internal error.
-	ERRSRV	ERRbaduid	The UID given (*smb_uid*) is not known as a valid ID on this LMX session.
-	SUCCESS	SUCCESS	Everything worked, no problems.

SMBunlink Preconditions

1. The SMB request, UID and TID are valid; the TID refers to a file system resource with write permissions.
2. *smb_path* refers to one or more existing files.
3. The directory containing the files to be unlinked must allow writes by the requesting process.
4. The files to be unlinked are not opened (except by the request process in compatibility mode).

SMBunlink Postconditions

The file's directory entries are removed.

SMBunlink Side Effects

None.

Conventions

- Access (see Section 4.3.2 on page 30).
- Attribute (see Section 4.3.1 on page 30).
- Filename (see Section 3.5 on page 15).
- Opportunistic Locking (see Section 3.8.2 on page 20).
- Wildcards (see Section 3.6 on page 17).

Chapter 8

Core SMB Directory and Attribute Operations

This section defines the elements of the core SMB protocol which manipulate directories and attributes. They are:

SMBmkdir	create an empty directory
SMBrmdir	delete an empty directory
SMBsearch	perform a wildcard lookup in a directory
SMBgetatr	get file attributes
SMBsetatr	set file attributes
SMBdskattr	get information about the LMX server's file system
SMBchkpath	ensure a path is valid and points to a directory

8.1 SMBmkdir Specification

SMBmkdir Detailed Description

This core protocol request creates a new directory which must not already exist. Write permission is required in the specified directory's parent directory.

SMBmkdir Deviations

The LMX server obeys the rules for mapping the new directory into the CAE file system (refer to Section 4.3.1 on page 30).

SMBmkdir Field Descriptions

From SMB redirector		To SMB redirector	
Field Name	Field Value	Field Name	Field Value
smb_com	*SMBmkdir*	*smb_com*	*SMBmkdir*
smb_wct	0	*smb_wct*	0
smb_bcc	min=2	*smb_bcc*	0
smb_buf[]	*smb_path*		

smb_path An ASCIIZ (type 04) buffer containing the name of the directory to be created.

SMBmkdir Error Code Descriptions

CAE Code	DOS Class	DOS Code	Description
ENOTDIR	ERRDOS	ERRbadpath	A component of the path-prefix was not a directory.
ENOENT	ERRDOS	ERRbadpath	A component of the path-prefix did not exist.
EACCES	ERRDOS	ERRnoaccess	A component of the path-prefix denied search permission.
EROFS	ERRHRD	ERRnowrite	Attempt to write a read-only file system.
EEXIST	ERRDOS	ERRfilexists	The specified path already exists.
ENOSPC	ERRDOS	ERRnoaccess	The parent's directory is full.
EIO	ERRHRD	ERRdata	Physical I/O error on disk.
EMLINK	ERRDOS	ERRnoaccess	Too many links to the parent directory.
-	ERRSRV	ERRerror	Internal error.
-	ERRSRV	ERRbaduid	The UID given (*smb_uid*) is not known as a valid ID on this LMX session.
-	SUCCESS	SUCCESS	Everything worked, no problems.

SMBmkdir Preconditions

1. Valid SMB request, UID and TID; TID is for a file system subtree.
2. The parent directory of the new directory must have the necessary access rights to create a directory.

SMBmkdir Postconditions

The directory is created in the file system.

SMBmkdir Side Effects

None.

Conventions

- Filename (see Section 3.5 on page 15).

8.2 SMBrmdir Specification

SMBrmdir Detailed Description

This core protocol request deletes an empty directory. The requesting UID must have write permission in the target directory's parent directory.

Because an LMX server may serve multiple requests on the same resource simultaneously, there may be interactions between the execution of this request and ongoing searches of the same resource (*SMBsearch*, *SMBffirst*, *SMBfunique*, *SMBfclose*). Although there is no prohibition on deleting directories actively being searched, an LMX server may cause the search to appear to have reached the end of the directory since no more entries will be found.

SMBrmdir Deviations

None.

SMBrmdir Field Descriptions

From SMB redirector		To SMB redirector	
Field Name	Field Value	Field Name	Field Value
smb_com	*SMBrmdir*	*smb_com*	*SMBrmdir*
smb_wct	0	*smb_wct*	0
smb_bcc	min=2	*smb_bcc*	0
smb_buf[]	*smb_path*		

smb_path An ASCIIZ (type 04) buffer containing the name of the directory to delete.

SMBrmdir Error Code Descriptions

CAE Code	DOS Class	DOS Code	Description
ENOTDIR	ERRDOS	ERRbadpath	A component in the path-prefix is not a directory.
ENOENT	ERRDOS	ERRbadfile	The specified directory does not exist.
EACCES	ERRDOS	ERRnoaccess	A component in the path denies the required permission.
EROFS	ERRHRD	ERRnowrite	Attempt to modify a read-only file system.
EBUSY	ERRDOS	ERRnoaccess	The directory is in use and cannot be removed at this time.
EEXIST	ERRDOS	ERRnoaccess	Attempt to remove a non-empty directory.
-	ERRSRV	ERRerror	Internal error.
-	ERRSRV	ERRbaduid	The UID given (*smb_uid*) is not known as a valid ID on this LMX session.
-	SUCCESS	SUCCESS	Everything worked, no problems.

SMBrmdir Preconditions

1. Valid SMB request, UID and TID; TID refers to a file system subtree.
2. The UID has write access to the parent directory of the target.

SMBrmdir Postconditions

The directory is deleted.

SMBrmdir Side Effects

An in-progress search from another process may receive an inconsistent view of the resource.

Conventions

- Access (see Section 4.3.2 on page 30).
- Filename (see Section 3.5 on page 15).

8.3 SMBsearch Specification

SMBsearch Detailed Description

This core protocol request searches a directory for one or more regular files matching a wildcard template. Two forms of the *SMBsearch* request exist: *SearchFirst* and *SearchNext*.

Every search begins when an SMB redirector sends a *SearchFirst* request to the LMX server asking for *n* files that match a specified wildcard template. The LMX server sends a response containing the directory information for up to *n* files found which match the template. The response contains a search handle defined below.

The SMB redirector may then resume the search at any search handle of a previous *SMBsearch* response. The LMX server responds to *SearchNext* with the directory information for up to *n* additional matching files, picking up from the point indicated by the search handle.

The SMB redirector does not indicate when a search is complete; that is, there is no *SearchDone* request.

SMBsearch Deviations

Since the SMB redirector never closes a search, the LMX server must use some heuristics in determining when to release resources associated with a search. These heuristics should never result in a search being declared terminated by the LMX server while it is still possible for the SMB redirector to continue it. Some possible heuristics are:

1. An *SMBexit* request from the same process is received.
2. The TID containing the search is broken.
3. The LMX session containing the search times out.
4. An error of any sort is returned in response to an *SMBsearch* request.

For the root directory of the directory subtree located by the TID the directory entries . and .. are not returned to the SMB redirector. If a volume label is returned it should be a printable string. Some SMB redirector applications will print this string, but no other semantics are associated with it.

The system, archive and hidden bits of the file attribute fields are treated in accordance with the Attribute convention (see Section 4.3.1 on page 30).

An LMX server must guarantee never to return information on a given file twice in the same *SMBsearch* sequence, provided *find_buf_search_id* contents are not reused by the SMB redirector. Some CAE systems can rearrange the information within a directory without the LMX server's knowledge; for example, entries may be moved around to pack a directory, etc. Because of this, LMX servers may not be able to guarantee that all files are reported once; that is, some files matching *smb_pathname* and *smb_attr* may not be reported to the SMB redirector.

SMBsearch Field Descriptions

Request Format:

From SMB redirector		To SMB redirector	
Field Name	Field Value	Field Name	Field Value
smb_com	*SMBsearch*	*smb_com*	*SMBsearch*
smb_wct	2	*smb_wct*	1
smb_vwv[0]	*smb_count*	*smb_vwv*[0]	*smb_count*
smb_vwv[1]	*smb_attr*	*smb_bcc*	min=3
smb_bcc	min=5		*smb_data*
	smb_pathname		
	smb_search_id[]		

smb_count A signed integer. In the request, the maximum number of entries to find and return in the response (*n*); in the response, the number of entries actually returned. If no matching entries were found between the point where this particular *SearchFirst* or *SearchNext* began, a zero (0) should be returned. The number of entries returned will be the minimum of:

— the number of entries requested

— the number of (complete) entries that will fit in the negotiated SMB buffer

— the number of entries that match the requested name pattern and attributes

smb_attr An attribute field. If supported, the LMX server will only return directory entries whose attributes match this field as well as the wildcard pathname. Unless this field specifies the volume label, normal files whose names match the wildcard are always returned. If this field specifies the volume label, only the volume label information is returned.

smb_pathname An ASCIIZ (type 04) buffer containing the wildcard path to search. Only the last component of the pathname may contain a wildcard.

smb_search_id A Variable Block (type 05), 21 or 0 bytes in length. If this is a zero-byte Data Block, it is a *SearchFirst* request; otherwise it is a *SearchNext* request containing the *find_buf_search_id* (see below) returned in the last *dir_info* structure in a previous *SearchFirst* or *SearchNext* response.

smb_data A Variable Block (type 05) containing an array of *dir_info* structures, tightly packed. The total size of the array is 43**smb_count*.

The *dir_info* structure contains information about each file which matched the wildcard *smb_pathname* (and, optionally, the *smb_attr* attributes). The structure contains:

Position	Field Name	Description
00	*find_buf_search_id*	A 21-byte string whose structure is defined below.
21	*find_buf_attr*	The attribute field for the file.
23	*find_buf_time*	A 16-bit time field, indicating the time of last modification.
25	*find_buf_date*	A 16-bit date field, indicating the date of last modification.
27	*find_buf_size*	A 32-bit integer giving the size of the file.
31	*find_buf_pname*	A blank-padded string, 13 characters in length, giving the name of the file in printable form. For example, AB.Tx would be encoded as AB.Tx vvvvvvvv. (v is a blank space.)

The *find_buf_search_id* referred to as the search handle above appears in two places: in the *SearchNext* request, and at the beginning of each *dir_info* structure. It contains state information the LMX server needs to continue a search. Its structure is as follows:

Position	Field Name	Description
00	*sr_res1*	Reserved for SMB redirector use. This field must be maintained by the LMX server. In other words, the value specified by the SMB redirector system must be returned in the appropriate search handle of the response.
01	*sr_servdata*	16-byte field reserved for LMX server use. Usually maintains state to continue searches; see paragraph below.
17	*sr_res2*[4]	4-byte field reserved for SMB redirector use. This field must be maintained by the LMX server in the same manner as the *sr_res1* field.

DOS SMB redirectors using the dialects PC NETWORK PROGRAM 1.0, MICROSOFT NETWORKS 1.03 and MICROSOFT NETWORKS 3.0 used the *sr_servdata* field in order to enhance the performance of the search sequence. If those SMB redirectors exist on the network, then the *sr_servdata* field is defined and the LMX server must maintain the following structure of information:

Position	Description
0-10	A compressed 11-byte string maintaining the search pattern for the directory search. This will include any meta-characters for the search. The . in DOS filenames (preceding the 3-byte filename extension) is assumed, in that it is not maintained in the string but rather inserted prior to the last 3 characters of the field. The first 8 characters are blank padded unless meta-characters are used. In the case of meta-characters, a * is expanded out into the appropriate number of question marks.
11	An unsigned byte. No assumptions are made on this value except that it should be non-zero.
12-13	An unsigned 16-bit integer which maintains the directory index value for this search entry. This value starts counting from zero and continues in a linear sequence. Some SMB redirectors are known to modify this value to allow them to resume a directory search at an arbitrary location.
14-15	An unsigned 16-bit integer that may be used by the LMX server. It should not be zero.

SMBsearch Error Code Descriptions

CAE Code	Error DOS Class	Error DOS Code	Description
EACCES	ERRDOS	ERRnoaccess	No permission for the specified pathname.
EIO	ERRHRD	ERRdata	Physical I/O error on disk.
EMFILE	ERRSRV	ERRnoresource	Exhausted process file handle supply.
ENFILE	ERRSRV	ERRnoresource	Exhausted system file handle supply.
ENOENT	SUCCESS	SUCCESS	Ignored (a file disappeared or didn't exist).
ENOTDIR	ERRDOS	ERRbadpath	Component in pathname was not a directory.
EOF	ERRDOS	ERRnofiles	Search can find no more files.
-	ERRSRV	ERRerror	LMX server internal error.
-	ERRDOS	ERRbadfid	*search_id* was not active.
-	ERRSRV	ERRbaduid	The UID given (*smb_uid*) is not known as a valid ID on this LMX session.
-	SUCCESS	SUCCESS	Everything worked, no problems.

SMBsearch Preconditions

1. Valid SMB, UID and TID; the TID refers to a file system subtree.
2. The UID has appropriate permission on all directories in *smb_pathname*.
3. The LMX server has not declared the search terminated.

SMBsearch Postconditions

1. After a *SearchFirst* request, the various directories under search are opened as necessary, and sufficient state is maintained to continue the search.
2. After a *SearchNext*, the retained state information is updated to permit continuing the search without returning *dir_info* on the same file twice.

SMBsearch Side Effects

Various directories are open for reading as long as the search is active. This may delay other requests from other SMB redirectors (for example, *SMBrmdir*).

Conventions

- Access (see Section 4.3.2 on page 30).
- Attribute (see Section 4.3.1 on page 30).
- Filename (see Section 3.5 on page 15).
- Wildcard (see Section 3.6 on page 17).

8.4 SMBgetatr Specification

SMBgetatr Detailed Description

This core protocol request is used to obtain information about a regular file or directory.

SMBgetatr Deviations

1. The archive, system and hidden file attribute bits are treated according to the attribute mapping convention.
2. The *smb_time* value returned will be the file's last modified time (as set by a previous close operation).

SMBgetatr Field Descriptions

From SMB redirector		To SMB redirector	
Field Name	Field Value	Field Name	Field Value
smb_com	*SMBgetatr*	*smb_com*	*SMBgetatr*
smb_wct	0	*smb_wct*	10
smb_bcc	min=2	*smb_vwv*[0]	*smb_attr*
smb_buf[]	*smb_path*	*smb_vwv*[1-2]	*smb_time*
		smb_vwv[3-4]	*smb_size*
		smb_vwv[5-9]	reserved (MBZ)
		smb_bcc	0

smb_path An ASCIIZ (type 04) buffer containing the name of the regular file or directory for which information is requested.

smb_attr A 16-bit attribute field describing the file.

smb_time A 32-bit time giving the last modify time for the file.

smb_size A 32-bit integer containing the current size of the file in bytes.

SMBgetatr Error Code Descriptions

CAE Code	DOS Class	DOS Code	Description
EACCES	ERRDOS	ERRnoaccess	Component of path-prefix denies search permission.
EINTR	ERRSRV	ERRerror	A signal was caught during some system call.
ENOENT	ERRDOS	ERRbadfile	File does not exist, or component of pathname does not exist.
ENOTDIR	ERRDOS	ERRbadpath	Component of path-prefix is not a directory.
-	ERRDOS	ERRnoaccess	Read permission required.
-	ERRSRV	ERRinvtid	TID specified in command is invalid.
-	ERRSRV	ERRinvdevice	Invalid resource type: TID was not for a file system subtree.
-	ERRSRV	ERRbaduid	The UID given (*smb_uid*) is not known as a valid ID on this LMX session.
-	SUCCESS	SUCCESS	Everything worked, no problems.

SMBgetatr Preconditions

1. The SMB redirector has the appropriate permission to the file system subtree.
2. *smb_path* refers to an existing file or directory.

SMBgetatr Postconditions

The *smb_attr* and *smb_time* fields are accurate for files and directories; *smb_size* is correct only for files and is meaningless for directories.

SMBgetatr Side Effects

None.

Conventions

- Attribute (see Section 4.3.1 on page 30).
- Filename (see Section 3.5 on page 15).

8.5 SMBsetatr Specification

SMBsetatr Detailed Description

This core protocol request is used to set information about an existing regular file or directory.

SMBsetatr Deviations

1. The archive, system and hidden file attribute bits are treated according to the file attributes conventions. Reference Section 4.3.1 on page 30 for additional information on file attribute handling.
2. The *smb_time* specified will become the last modify time for the file.

SMBsetatr Field Descriptions

From SMB redirector		To SMB redirector	
Field Name	Field Value	Field Name	Field Value
smb_com	*SMBsetatr*	*smb_com*	*SMBsetatr*
smb_wct	8	*smb_wct*	0
smb_vwv[0]	*smb_attr*	*smb_bcc*	0
smb_vwv[1-2]	*smb_time*		
smb_vwv[3-7]	reserved (MBZ)		
smb_bcc	min=2		
smb_buf[]	*smb_path*		
	smb_nul		

smb_attr A file attribute field, to be given to the file (see Section 3.5 on page 15 for details of the Attribute convention).

smb_time A 32-bit time giving the last modify time for the file. A value of 0 indicates the last modify time should be unchanged.

smb_path An ASCIIZ (type 04) buffer containing the name of the regular file or directory for which information is to be set.

smb_nul An ASCIIZ (type 04) buffer containing the null string.

SMBsetatr Error Code Descriptions

CAE Code	DOS Class	DOS Code	Description
EACCES	ERRDOS	ERRnoaccess	Search permission is denied on a component of the path-prefix.
EACCES	ERRSRV	ERRaccess	The UID does not have appropriate privilege and is not the owner of the file and the read-only attribute flag was changed.
EINTR	ERRSRV	ERRerror	A signal was caught during the system call.
ENOENT	ERRDOS	ERRbadfile	File does not exist, or component of pathname does not exist.
ENOTDIR	ERRDOS	ERRbadpath	Component of path-prefix is not a directory.
EPERM	ERRSRV	ERRaccess	The UID does not have appropriate privilege and is not the owner of the file and time is non-zero.
EROFS	ERRSRV	ERRaccess	The file system containing the file is read-only.
-	ERRSRV	ERRinvnid	TID specified in command is invalid.
-	ERRSRV	ERRinvdevice	The TID does not refer to a file system subtree.
-	ERRSRV	ERRbaduid	The UID given (*smb_uid*) is not known as a valid ID on this LMX session.
-	SUCCESS	SUCCESS	Everything worked, no problems.

SMBsetatr Preconditions

1. The SMB redirector has sent a valid SMB request with a valid UID and a valid TID for a file system subtree.
2. *smb_path* refers to an existing file or directory.
3. The specified UID or TID represents appropriate privilege to perform the action.

SMBsetatr Postconditions

The file attribute and time will be set accordingly, or an error will be returned.

SMBsetatr Side Effects

1. If the read-only attribute was changed, the access mode for the file will have been changed accordingly. For example, when the read-only attribute is removed the LMX server will set those write permission bits for a file not explicitly masked out by the current *umask* value.
2. The last modify time for the file will be changed if the specified time was non-zero.

Conventions

- Access (see Section 4.3.2 on page 30).
- Attribute (see Section 4.3.1 on page 30).
- Filename (see Section 3.5 on page 15).

8.6 SMBdskattr Specification

SMBdskattr Detailed Description

This core protocol request returns some information on the resource's associated file system subtree.

SMBdskattr Deviations

An LMX server may return zero (0) in the *smb_vwv*[4] (media identifier code) field.

SMBdskattr Field Descriptions

From SMB redirector		To SMB redirector	
Field Name	Field Value	Field Name	Field Value
smb_com	*SMBdskattr*	*smb_com*	*SMBdskattr*
smb_wct	0	*smb_wct*	5
smb_bcc	0	*smb_vwv*[0]	number of allocation units/server
		smb_vwv[1]	number of blocks/allocation unit
		smb_vwv[2]	block size (in bytes)
		smb_vwv[3]	number of free allocation units
		smb_vwv[4]	reserved (media identifier code)
		smb_bcc	0

SMBdskattr Error Code Descriptions

CAE Code	DOS Class	DOS Code	Description
ENOENT	ERRHRD	ERRnotready	The file system has been removed from the system.
ENOTDIR	ERRHRD	ERRnotready	The file system has been removed from the system.
EIO	ERRHRD	ERRdata	Physical I/O error on disk.
-	ERRSRV	ERRaccess	Read permission is required.
-	ERRSRV	ERRinvtid	Invalid TID specified.
-	ERRSRV	ERRinvdevice	Invalid resource type (that is, no file system subtree) specified.
-	ERRSRV	ERRerror	Other CAE and internal errors.
-	ERRSRV	ERRbaduid	The UID given (*smb_uid*) is not known as a valid ID on this LMX session.
-	SUCCESS	SUCCESS	Everything worked, no problems.

SMBdskattr Preconditions

The SMB request, UID and TID must be valid and represent the appropriate access rights to perform the action.

SMBdskattr Postconditions

None.

SMBdskattr Side Effects

None.

Conventions

- File System Issues (see Section 4.3.3 on page 30).

8.7 SMBchkpath Specification

SMBchkpath Detailed Description

This core protocol request verifies that a path exists and is a directory. For example, SMB redirectors which maintain a concept of a working directory might use *SMBchkpath* to verify the validity of a change working directory command. Note that an LMX server does not have a concept of working directory. The SMB redirector must always supply a full pathname (relative to the TID).

SMBchkpath Deviations

None.

SMBchkpath Field Descriptions

From SMB redirector		To SMB redirector	
Field Name	Field Value	Field Name	Field Value
smb_com	*SMBchkpath*	*smb_com*	*SMBchkpath*
smb_wct	0	*smb_wct*	0
smb_bcc	min=2	*smb_bcc*	0
smb_buf[]	*smb_path*		

smb_path An ASCIIZ (type 04) buffer containing the name of the directory to be checked.

SMBchkpath Error Code Descriptions

CAE Code	DOS Class	DOS Code	Description
ENOTDIR	ERRDOS	ERRbadpath	A component of the path was not a directory.
ENOENT	ERRDOS	ERRbadfile	The specified directory does not exist.
EACCES	ERRDOS	ERRnoaccess	A component of the path lacked search permission.
EACCES	ERRSRV	ERRaccess	No read permission in specified directory.
ENXIO	ERRDOS	ERRbadpath	The specified path wasn't a directory.
ENFILE	ERRDOS	ERRnofids	System file table full.
EMFILE	ERRDOS	ERRnofids	LMX session has too many open files.
EIO	ERRHRD	ERRdata	Physical I/O error on disk.
-	ERRSRV	ERRinvnid	Invalid TID specified.
-	ERRSRV	ERRerror	Internal error.
-	ERRSRV	ERRbaduid	The UID given (*smb_uid*) is not known as a valid ID on this LMX session.
-	SUCCESS	SUCCESS	Everything worked, no problems.

SMBchkpath Preconditions

SMB request, UID and TID are valid and represent the appropriate access rights to perform the action.

SMBchkpath Postconditions

If no error is returned, *smb_path* referred to a valid existing directory which is readable by the SMB redirector.

SMBchkpath Side Effects

None.

Conventions

- Filename (see Section 3.5 on page 15).

Chapter 9

Core SMB Spool Operation Requests

This section defines the elements of core SMB protocol which support spooling and printing operations. They are:

SMBsplopen	create a new spool file
SMBsplwr	write to a spool file
SMBsplclose	close a spool file and queue it for spooling
SMBsplretq	return information on the spool queue

9.1 SMBsplopen Specification

SMBsplopen Detailed Description

This core protocol request will create a spool file. The file will be deleted once it has been printed. The LMX server will grant write permission to the creator of the file. No other LMX session will be given any access permissions to the file.

All users will have read permission on the print spool queue, but only the print LMX server has write permission to it.

SMBsplopen Deviations

Some LMX servers do not distinguish between text and graphics modes.

SMBsplopen Field Descriptions

From SMB redirector		To SMB redirector	
Field Name	Field Value	Field Name	Field Value
smb_com	*SMBsplopen*	*smb_com*	*SMBsplopen*
smb_wct	2	*smb_wct*	1
smb_vwv[0]	*smb_psdlen*	*smb_vwv*[0]	*smb_fid*
smb_vwv[1]	*smb_mode*	*smb_bcc*	0
smb_bcc	min = 2		
smb_buf	*smb_ident*		

smb_psdlen A 16-bit integer giving the length of printer setup data to be sent. This means that the first *smb_psdlen* bytes of data sent to this spool file will be treated by the LMX server as setup data.

smb_mode A 16-bit field providing additional control over the printing of this file. The field can have the following values:

- 0 Text mode. Some LMX servers expand ASCII TABs to spaces in this mode.
- 1 Graphics mode. The LMX server treats the data as raw octets and will not interpret or change it.

smb_ident An ASCIIZ (type 04) buffer containing a suggested name for the spool file. The LMX server may ignore, truncate, or otherwise use this information in any way.

smb_fid The FID of the spool file. Data written to this FID will be spooled.

SMBsplopen Error Code Descriptions

CAE Code	DOS Class	DOS Code	Description
-	ERRSRV	ERRerror	The request SMB was invalid or malformed.
-	ERRSRV	ERRerror	The LMX server cannot find the spool queue for this file.
-	ERRSRV	ERRqfull	Insufficient resources to create the print job.
-	ERRSRV	ERRqtoobig	The queue is full: no entry is available to create the job.
-	ERRSRV	ERRerror	The LMX server has exhausted some resource and cannot create the print job.
EACCES	ERRDOS	ERRnoaccess	Search permission is denied on a component of the path-prefix.
EINTR	ERRSRV	ERRerror	A signal was caught during a system call.
EMFILE	ERRDOS	ERRnofids	Maximum number of file descriptors are currently open in this process.
ENFILE	ERRDOS	ERRnofids	System file table is full.
EROFS	ERRSRV	ERRerror	The spool file or spool queue resides on a read-only file system.
-	ERRSRV	ERRinvdevice	The TID does not refer to a printer resource.
-	ERRSRV	ERRbaduid	The UID given (*smb_uid*) is not known as a valid ID on this LMX session.
-	SUCCESS	SUCCESS	Everything worked, no problems.

SMBsplopen Preconditions

The SMB request, UID and TID are valid and represent the appropriate access rights for the action.

SMBsplopen Postconditions

1. If successful, *smb_fid* contains the FID to be used in subsequent *SMBsplwr* requests for this spool file.
2. Although some resources were reserved to create the spool file, there is no guarantee that sufficient resources exist for a given amount of data to be spooled within this spool file.

SMBsplopen Side Effects

A spool file has been created on the LMX server.

Conventions

- Print Spooling (see Section 4.6 on page 35).

9.2 SMBsplwr Specification

SMBsplwr Detailed Description

This core protocol request appends the data block to the spool file specified by the FID. The first block sent to a spool file must contain the printer setup data; the length of this data was specified in the *SMBsplopen* request. Additional data may appear with the first block sent.

SMBsplwr Deviations

It is possible that LMX servers are such that if an *SMBsplwr* request contained a message of length greater than the maximum transmit size for the TID specified, the LMX server would abort the LMX session to the SMB redirector (see Section 6.1 on page 55 and Section 6.2 on page 57). Rather than aborting, the LMX server could accept an amount of data which is the lesser of the amount the SMB redirector indicated would be sent and the size of the data in the buffer.

SMBsplwr Field Descriptions

From SMB redirector		To SMB redirector	
Field Name	Field Value	Field Name	Field Value
smb_com	*SMBsplwr*	*smb_com*	*SMBsplwr*
smb_wct	1	*smb_wct*	0
smb_vwv[0]	*smb_fid*	*smb_bcc*	0
smb_bcc	min=4		
smb_buf	*smb_data*		

smb_fid The FID for a spool file. Obtained in an *SMBsplopen* response.

smb_data A Data Block (type 01) buffer, containing data to be written to the spool file. The first bytes of the first *smb_data* field sent to a newly-opened spool file are considered to be printer setup data; the length of this setup data is specified in the *smb_psdlen* field of the *SMBsplopen* request.

SMBsplwr Error Code Descriptions

CAE Code	DOS Class	DOS Code	Description
EBADF	ERRDOS	ERRbadfid	FID is valid, but no longer accepted by the underlying operating system.
-	ERRDOS	ERRbadfid	Invalid FID.
EAGAIN	ERRDOS	ERRnoaccess	A temporary resource limitation prevented this data from being written.
EIO	ERRHRD	ERRwrite	A physical I/O error has occurred.
-	ERRSRV	ERRqtoobig	A part of the spooler subsystem failed due to lack of file system space.
-	ERRSRV	ERRinvnid	The TID in the command is invalid.
-	ERRSRV	ERRbaduid	The UID given (*smb_uid*) is not known as a valid ID on this LMX session.
-	SUCCESS	SUCCESS	Everything worked, no problems.

SMBsplwr Preconditions

1. The SMB request, UID and TID are valid and represent the appropriate access rights for the action.
2. The spool file specified by *smb_fid* must have been opened with *SMBsplopen*.

SMBsplwr Postconditions

If no error is returned, the data sent in the request will be written to the spool file.

SMBsplwr Side Effects

None.

Conventions

- Print Spooling (see Section 4.6 on page 35).

9.3 SMBsplclose Specification

SMBsplclose Detailed Description

This core protocol request invalidates the specified FID and queues the file for spooling. The FID must reference a spool file.

SMBsplclose Deviations

None.

SMBsplclose Field Descriptions

From SMB redirector		To SMB redirector	
Field Name	Field Value	Field Name	Field Value
smb_com	*SMBsplclose*	*smb_com*	*SMBsplclose*
smb_wct	1	*smb_wct*	0
smb_vwv[0]	*smb_fid*	*smb_bcc*	0
smb_bcc	0		

smb_fid The FID of the spool file to be closed and queued for spooling.

SMBsplclose Error Code Descriptions

CAE Code	DOS Class	DOS Code	Description
EBADF	ERRSRV	ERRerror	The LMX server could not use a valid FID.
-	ERRDOS	ERRbadfid	The FID in the request is not valid.
-	ERRSRV	ERRinvdevice	The FID does not refer to an open spool file.
-	ERRSRV	ERRinvtid	The TID in the command is invalid.
-	ERRSRV	ERRbaduid	The UID given (*smb_uid*) is not known as a valid ID on this LMX session.
-	SUCCESS	SUCCESS	Everything worked, no problems.

SMBsplclose Preconditions

1. The SMB request, UID and TID are valid and represent the appropriate access rights for the action.
2. *smb_fid* must refer to a spool file opened with *SMBsplopen*.

SMBsplclose Postconditions

1. If no errors have occurred, the spool file will be closed and the job scheduled.
2. If an error has occurred, it is possible that the data was not printed and may have been lost.

SMBsplclose Side Effects

1. The data is spooled. Refer to Section 4.6 on page 35.
2. During or after the printing of the file, the resources consumed by it will be released.

Conventions

- Print Spooling (see Section 4.6 on page 35).

9.4 SMBsplretq Specification

SMBsplretq Detailed Description

This core protocol request obtains a list of the elements currently in the print spool queue on the LMX server. Zero or less than the requested number of elements will be returned only when the beginning or end of the queue is encountered.

SMBsplretq Deviations

Some LMX servers cannot search the queue backwards, and will respond to requests for backward searches with a forward search instead. The in intercept bit in the *smb_status* field of *smb_data* will never be used.

SMBsplretq Field Descriptions

From SMB redirector		To SMB redirector	
Field Name	Field Value	Field Name	Field Value
smb_com	*SMBsplretq*	*smb_com*	*SMBsplretq*
smb_wct	2	*smb_wct*	2
smb_vwv[0]	*smb_maxcount*	*smb_vwv*[0]	*smb_count*
smb_vwv[1]	*smb_st_index*	*smb_vwv*[1]	*smb_res_index*
smb_bcc	0	*smb_bcc*	min = 3
		smb_buf	*smb_data*

smb_maxcount A 16-bit integer specifying the maximum number of entries to return. If positive, search forward in the queue; if negative, search backwards. If *smb_maxcount* entries require more data than can fit in a message, those entries which fit are returned and no error is generated.

smb_st_index A 16-bit integer indicating the first entry in the queue to return. A value of 0 indicates the start of the queue; other values should only come from the *smb_res_index* field of previous *SMBsplretq* responses.

smb_count A 16-bit integer indicating how many entries were actually returned.

smb_res_index A 16-bit integer giving the index of the entry following the last entry returned; it may be used as the start index in a subsequent request to resume the queue listing.

smb_data A Data Block (type 01) buffer containing an array of *smb_count* queue element structures. Each queue element is 28 bytes in length and contains the following fields:

00	16-bit field	*smb_date*
02	16-bit field	*smb_time*
04	8-bit field	*smb_status*
05	16-bit field	*smb_file*
07	32-bit field	*smb_size*
11	8-bit field	*smb_res*
12	8-bit field	*smb_name*[16]

smb_date A 16-bit field containing the date for when the file was created. Refer to Section 5.3.2 on page 43

smb_time A 16-bit field telling time for when the file was created. Refer to Section 5.3.1 on page 43.

smb_status An 8-bit field indicating the file's status in the print spool queue as follows:

0x01	held or stopped
0x02	printing
0x03	awaiting print
0x04	in intercept (never used)
0x05	file had error
0x06	printer error
0x07-0xff	reserved; do not use

smb_file A 16-bit integer containing the spool job ID, as generated on the LMX server during the processing of the *SMBsplopen* request for this spool file.

smb_size A 32-bit integer containing the size of the file in bytes.

smb_res An 8-bit reserved field; MBZ (Must Be Zero).

smb_name A 16-byte string identifying the spool file. This may be the originating SMB redirector's name or the spool filename. The spool filename is created by the LMX server when an *SMBsplopen* request is received. This string is left-justified and NULL-filled in the field.

SMBsplretq Error Code Descriptions

CAE Code	DOS Class	DOS Code	Description
-	ERRHRD	ERRnotready	Any of several errors could be mapped to this error code.
-	ERRHRD	ERRerror	A resource limitation was exceeded.
-	ERRSRV	ERRbaduid	The UID given (*smb_uid*) is not known as a valid ID on this LMX session.
-	SUCCESS	SUCCESS	Everything worked, no problems.

SMBsplretq Preconditions

1. The maximum SMB size permits at least 28* *smb_max_count* bytes of data in addition to the SMB header and request subheader.

SMBsplretq Postconditions

None.

SMBsplretq Side Effects

None.

Conventions

This is a request where the UID and the TID need not be valid for service.

- Print Spooling (see Section 4.6 on page 35).

Chapter 10

Core Plus SMB File Operations

This section defines the elements of the core plus SMB protocol which provide for file operations. They are:

SMBnegprot	negotiate modifications when the core plus dialect is selected by the LMX server
SMBreadbmpx	read block multiplexed
SMBwritebmpx	write block multiplexed
SMBreadbraw	read block raw from a file
SMBwritebraw	write block raw to a file
SMBlockread	lock a byte range and read it
SMBwriteunlock	write to a byte range and unlock it
SMBwriteclose	write to a file and close it

10.1 SMBnegprot Specification

SMBnegprot Detailed Description

This SMB protocol request is sent to establish the protocol dialect that the SMB redirector and LMX server will use when communicating with each other. The SMB redirector sends a list of dialects that it can use for communication. The LMX server responds with a selection of one of those dialects (numbered 0 to *n*) or −1 indicating that none of the dialects were acceptable. Exactly one negotiate message must be sent on each NetBIOS session; subsequent negotiate requests must be rejected with an error response and no action will be taken. The rules for the use of *SMBnegprot* outlined in Section 6.1 on page 55 hold here as well.

SMBnegprot Deviations

None.

SMBnegprot Field Descriptions

Field descriptions for other dialects of the SMB protocol (*SMBnegprot*) are:

From SMB redirector		To SMB redirector	
Field Name	Field Value	Field Name	Field Value
smb_com	*SMBnegprot*	*smb_com*	*SMBnegprot*
smb_wct	0	*smb_wct*	13
smb_bcc	min=2	*smb_vwv*[0]	*smb_index*
smb_buf[ ]	dialect0	*smb_vwv*[1-4]	*smb_rsvd0*
	.	*smb_vwv*[5]	*smb_blkmode*
	.	*smb_vwv*[6-12]	*smb_rsvd1*
		smb_bcc	0

The fields are defined as:

dialect*n*	A Dialect (type 02) buffer containing the name of a dialect (refer to Section 5.4 on page 48).
smb_index	The dialect selected by the LMX server; corresponds to the *index*th dialect string in the request, where the first string is numbered 0.
smb_rsvd0	Reserved; MBZ (Must Be Zero).
smb_blkmode	Whether or not *SMBreadbraw* and *SMBwritebraw* are supported. Bit 0 If set, *SMBreadbraw* is supported. Bit 1 If set, *SMBbwritebraw* is supported. Bit 2-15 Reserved; Must Be Zero. Some SMB redirectors when negotiating the core plus dialect ignore these bits and assume both SMBs are acceptable.
smb_rsvd1	Reserved; MBZ (Must Be Zero).
smb_bcc	This area is ignored in the core plus dialect.

Note that bit 0 of the *smb_flg* field in the SMB header of the response will be interpreted by the SMB redirector to indicate support for *SMBlockread* and *SMBwriteunlock*.

SMBnegprot Error Code Descriptions

If any error occurs, the LMX server will return <ERRSRV, ERRerror>; otherwise, <SUCCESS, SUCCESS> will be returned.

SMBnegprot Preconditions

The SMB redirector attempting to negotiate a protocol must have established a NetBIOS session with the LMX server.

SMBnegprot Postconditions

The SMB redirector that negotiated this protocol must be able to handle all aspects of the SMB dialect negotiated.

SMBnegprot Side Effects

The LMX server will keep a record of which dialect the SMB redirector negotiated and will use only that dialect in conversations with the SMB redirector.

Coventions

None.

10.2 SMBreadbraw Specification

SMBreadbraw Detailed Description

The read block raw request is used to maximise the performance of reading a large block of data from a file on the LMX server. Any supported file type can be read via *SMBreadbraw*. Up to 65,535 bytes can be read in one request/response regardless of the maximum negotiated buffer size.

When the SMB redirector sends this request, it guarantees no other request will be issued on the same LMX session until the response is received from the LMX server. Given this guarantee, the LMX server responds by sending just the requested data in a single transport message. No header of any sort is generated. Because the entire response is sent as a single message, the SMB redirector can determine how much data was actually sent.

If the request is to read more data than is present in the file, the read response will be of the length actually read from the file. If the read begins at or after EOF, or some other error is encountered, a zero-length message is sent in response. An SMB redirector will send a read request other than *SMBreadbraw* to find out what happened, at which time an EOF indication or error is returned in the response to that request.

If an error should occur at the SMB redirector end, all data must be received and thrown away. The LMX server will not be informed.

SMBreadbraw Deviations

Support for the timeout field for file types other than named pipes is optional. If timeouts are not supported, all requests are treated as non-blocking.

SMBreadbraw Field Descriptions

From SMB redirector		To SMB redirector	
Field Name	Field Value	Field Name	Field Value
smb_com	*SMBreadbraw*	raw data	
smb_wct	8		
smb_vwv[0]	*smb_fid*		
smb_vwv[1-2]	*smb_offset*		
smb_vwv[3]	*smb_maxcnt*		
smb_vwv[4]	*smb_mincnt*		
smb_vwv[5,6]	*smb_timeout*		
smb_vwv[7]	*smb_rsvd*		
smb_bcc	0		

smb_fid The FID for the read.

smb_offset A 32-bit unsigned integer giving the offset into the file, in bytes, at which the read is to begin.

smb_maxcnt An unsigned 16-bit field indicating the number of bytes to be read.

smb_mincnt If a timeout is specified, this is the minimum number of bytes that must be read for the request to return before timing out.

smb_timeout A 32-bit integer giving the number of milliseconds to wait for at least *smb_mincnt* bytes of data to be read. A value of zero (0) indicates the read should not block. A timeout of −1 means the LMX server should wait indefinitely. A timeout of −2 indicates the default timeout for the named pipe

should be used.

smb_rsvd A 16-bit reserved field, which should be ignored.

The response contains no headers or other overhead, and is a single message containing the bytes that were read. A zero-length message indicates either *smb_offset* pointed beyond the current EOF or some other error occurred.

SMBreadbraw Error Code Descriptions

No errors may be returned in the response to this request. Instead, any errors are saved until the next request for this file, at which time they will be returned.

SMBreadbraw Preconditions

1. The SMB redirector has sent a valid SMB request with a valid TID for a readable resource.
2. The FID is valid and the process has read access.

SMBreadbraw Postconditions

The LMX server has returned to the SMB redirector either all of the requested raw data, all of the data up to the EOF, or a response with no data.

SMBreadbraw Side Effects

Since the LMX server is not allowed to return errors with this SMB request, a return of 0 bytes can indicate either EOF, file system read error, outstanding break or block, or that the LMX server is temporarily out of some required resource. In the case of a 0 byte return, the SMB redirector should follow up with an *SMBread* or *SMBreadmpx* request at which time the LMX server can return an error if necessary.

Conventions

- Locking (see Section 4.4 on page 33).

10.3 SMBwritebraw Specification

SMBwritebraw Detailed Description

The write block raw message exchange provides a high-performance mechanism for transferring large amounts of data to be written to a file on the LMX server. Any supported file type, including spool files, may be written with this exchange.

The *SMBwritebraw* exchange behaves much like an *SMBwritebmpx* exchange, except that instead of additional data being sent in secondary requests, all the additional data is sent in a single raw message; that is, the first segment of data is sent in the primary request, and the remainder in a single message with no SMB header or *SMBwritebraw* subheader.

If all the data to be written fits in the primary request, a zero-length secondary request is still sent; even if the secondary request is zero-length, a secondary response must be generated when write-through mode was specified.

If the LMX server is busy or otherwise unable to support the raw write of the remaining data, the data sent with the primary request is still written (to stable store if write-through mode was set). If any other error occurs, the data is discarded. In either case, an appropriate error is returned in a secondary response. A primary response is only sent if the primary request was satisfied with no errors and the LMX server is prepared for a raw message.

SMBwritebraw Deviations

The *smb_timeout* and *smb_remaining* fields will not be supported with I/O devices.

SMBwritebraw Field Descriptions

SMB redirectors using the core plus dialect of the SMB protocol use a slightly different form of the *SMBwritebraw* primary request, and expect a slightly modified primary response. Both forms are shown below.

Primary *SMBwritebraw* (core plus only):

From SMB redirector		To SMB redirector	
Field Name	Field Value	Field Name	Field Value
smb_com	*SMBwritebraw*	*smb_com*	*SMBwritebraw*
smb_wct	10	*smb_wct*	0
smb_vwv[0]	*smb_fid*	*smb_bcc*	0
smb_vwv[1]	*smb_tcount*		
smb_vwv[2]	*smb_rsvd*		
smb_vwv[3,4]	*smb_offset*		
smb_vwv[5,6]	*smb_timeout*		
smb_vwv[7]	*smb_wmode*		
smb_vwv[8,9]	*smb_rsvd*		
smb_bcc	min=0		
smb_buf[]	*smb_data*		

smb_fid The FID of the file to be written to.

smb_tcount An unsigned 16-bit field giving the total number of bytes that will be written to the file. This value must be correct in at least one of the requests in the exchange; in other requests, it may be an over-estimate.

smb_rsvd These fields are reserved and should be ignored by the LMX server.

smb_offset A 32-bit integer giving the position in the file at which the bytes in the request should be written.

smb_timeout A 32-bit integer giving the number of milliseconds the LMX server may block while trying to complete the write. This value is ignored for regular files. For I/O devices and named pipes (refer to the X/Open CAE Specification, IPC Mechanisms for SMB), the LMX server will wait this much time to complete the write. If *smb_timeout* is –1, the LMX server will wait indefinitely; if it is –2, the server will wait the default amount of time for the file. An LMX server may choose to treat all timeouts as 0, that is, do not block.

smb_wmode A 16-bit flag field controlling the write mode. If bit 0 is set, write-through mode is requested; the LMX server will write all data atomically and acknowledge the write with the secondary response. If clear, write-behind is permitted; the LMX server need not write atomically and need not report completion. If bit 1 is set, the LMX server should fill in the *smb_remaining* field in the primary response.

smb_data The actual data to be written. This is a string of bytes in no particular format.

Note that, in the core plus protocol dialect, there is no padding between the end of the *smb_vwv*[] block and the data to be written.

Secondary *SMBwritebraw*:

From SMB redirector		To SMB redirector	
Field Name	Field Value	Field Name	Field Value
raw data		*smb_com*	*SMBwritec*
		smb_wct	1
		smb_vwv[0]	*smb_count*
		smb_bcc	0

smb_count The total number of bytes written. If this is different from the smallest *smb_tcount* sent by the SMB redirector, some error occurred (for example, out of free space on the file system).

SMBwritebraw Error Code Descriptions

CAE Code	DOS Class	DOS Code	Description
-	ERRDOS	ERRbadfid	Invalid FID.
-	ERRDOS	ERRnoaccess	File opened in deny write mode, or write range overlaps a lock.
-	ERRDOS	ERRbadaccess	Invalid open mode for the attempted operation.
-	ERRSRV	ERRerror	Corrupt SMB.
-	ERRSRV	ERRinvnid	Invalid TID.
-	ERRSRV	ERRnoresource	The LMX server is temporarily out of a needed resource.
-	ERRSRV	ERRtimeout	Requested operation timed out.
-	ERRSRV	ERRuseMPX	Can't do raw mode at this time; use *SMBwritebmpx*.
-	ERRSRV	ERRuseSTD	Can't do raw mode at this time; use *SMBwrite* or *SMBwriteX*.
-	ERRSRV	ERRbaduid	The UID given (*smb_uid*) is not known as a valid ID on this LMX session.
-	SUCCESS	SUCCESS	Everything worked, no problems.

SMBwritebraw Preconditions

1. The primary SMB was valid and specified a valid TID for a writable resource.
2. The FID was valid, and the process had write access to the file.
3. Before sending the secondary message, the LMX server must have sent a primary response. The LMX server has been able to write the accompanying data to disk, allocated the needed memory for a buffer, and sent the response to the SMB redirector.

SMBwritebraw Postconditions

1. If write-through mode is set, a primary response or secondary response indicates the data in the primary response has been written to stable store (unless some error other than ERRuseSTD or ERRuseMPX was returned).
2. After a primary response is received, the LMX server is ready for a raw secondary message.

SMBwritebraw Side Effects

None.

Conventions

- Locking (see Section 4.4 on page 33).

10.4 SMBlockread Specification

SMBlockread Detailed Description

This lock and read protocol request has the effect of explicitly locking the bytes in the specified range and then reading them. The lock is maintained until explicitly released by the SMB redirector or the SMB redirector closes the file. Only the bytes actually read by this request are locked, not the bytes specified in the advisory *smb_countleft* field.

Support for this SMB is optional; an LMX server should set the appropriate bit in the *smb_flg* field of the *SMBnegprot* response (see Section 6.1 on page 55 for other dialects of the SMB protocol and Section 5.1 on page 37).

SMBlockread Deviations

None.

SMBlockread Field Descriptions

The request and response format are identical to that of *SMBread* (see Section 7.4 on page 73).

SMBlockread Error Code Descriptions

For a more complete description of the potential error codes resulting from this SMB message see Section 7.4 on page 73 and Section 7.7 on page 81.

CAE Code	DOS Class	DOS Code	Description
-	ERRDOS	ERRnoaccess	No read access to TID.
EBADF	ERRDOS	ERRbadfid	Invalid FID.
-	ERRDOS	ERRlock	The intended read range overlaps a lock held by another process.
EPERM	ERRDOS	ERRbadaccess	No read access for the file.
-	ERRSRV	ERRerror	Corrupt SMB.
-	ERRSRV	ERRinvdevice	TID is not for a file system subtree.
-	ERRSRV	ERRinvnid	Invalid TID.
-	ERRSRV	ERRbaduid	The UID given (*smb_uid*) is not known as a valid ID on this LMX session.
-	SUCCESS	SUCCESS	Everything worked, no problems.

SMBlockread Preconditions

1. The SMB redirector has sent a valid SMB with a valid TID for a readable file system resource.
2. The FID is valid, and the process has read access to the file.
3. The range of bytes to be read is not already locked by some other process.

SMBlockread Postconditions

1. The requested number of bytes (*smb_bytecount*) has been locked, read and returned, in that order.
2. The current file position is left after the bytes read.

SMBlockread Side Effects

1. Other SMB redirector processes will be unable to access the locked record until the SMB redirector holding the lock has released it or unless they are using the same FID.
2. The LMX server may have pre-read the remaining bytes (*smb_countleft - smb_bytecount*) to increase the performance of subsequent reads from the same process.

Conventions

- Locking (see Section 4.4 on page 33).

10.5 SMBwriteunlock Specification

SMBwriteunlock Detailed Description

This write and unlock protocol request has the effect of writing to a range of bytes and then unlocking them. This request is usually complementary to an earlier usage of *SMBlockread* on the same range of bytes. Only the range of bytes actually written to is unlocked, not the range specified in the advisory *smb_countleft* field. If an error occurs during the write, the byte range should not be unlocked.

Aside from the lack of special handling of zero-length writes, this request behaves in an identical fashion to a core protocol *SMBwrite* request followed by a core protocol *SMBunlock* request.

Support for this SMB is optional; an LMX server should set the appropriate bit in the *smb_flg* field of the *SMBnegprot* response (see Section 6.1 on page 55 for other dialects of SMB protocol and Section 5.1 on page 37).

SMBwriteunlock Deviations

See Section 7.5 on page 76 and Section 7.8 on page 83.

SMBwriteunlock Field Descriptions

The *SMBwriteunlock* request and response format are identical to those of *SMBwrite* (see Section 7.5 on page 76).

SMBwriteunlock Error Code Descriptions

For a list of other error codes generated during the handling of this SMB see Section 7.5 on page 76 and Section 7.8 on page 83.

CAE Code	DOS Class	DOS Code	Description
-	ERRDOS	ERRlock	The requested range was locked by a different process.
-	SUCCESS	SUCCESS	Everything worked, no problems.

SMBwriteunlock Preconditions

1. The SMB redirector has sent a valid SMB request with a TID for a writable file system subtree.
2. The FID must be valid and the process must have write access.
3. The write operation must succeed before the unlock operation is attempted.

SMBwriteunlock Postconditions

1. Either the write succeeded or an error was returned.
2. If the write succeeded, the byte range was unlocked.

SMBwriteunlock Side Effects

Same as for *SMBwrite* and *SMBunlock*.

Conventions

- Locking (see Section 4.4 on page 33).

10.6 SMBwriteclose Specification

SMBwriteclose Detailed Description

The write and close protocol request is used to first write the specified bytes and then close the file. Any supported file type, including spool files, may be specified in this request. This request behaves identically to an *SMBwrite* request followed by an *SMBclose* request. Any buffered data must be flushed to stable store or to the device before the response is sent.

SMBwriteclose Deviations

See Section 7.5 on page 76 and Section 12.6 on page 168 for details.

SMBwriteclose Field Descriptions

From SMB redirector		To SMB redirector	
Field Name	Field Value	Field Name	Field Value
smb_com	*SMBwriteclose*	*smb_com*	*SMBwriteclose*
smb_wct	(6 or 12)	*smb_wct*	1
smb_vwv[0]	*smb_fid*	*smb_vwv*[0]	*smb_count*
smb_vwv[1]	*smb_count*	*smb_bcc*	0
smb_vwv[2-3]	*smb_offset*		
smb_vwv[4-5]	*smb_time*		
smb_vwv[6-11]	*smb_rsvd*		
smb_bcc	(1+ *smb_count*)		
smb_buf[]	*smb_pad*		
	smb_data		

smb_fid The FID to be closed.

smb_count In the request, the number of bytes of data to be written. In the response, the number of bytes that were actually written.

smb_offset A 32-bit offset into the file, in bytes, at which the data is to be written.

smb_time A 32-bit time value to be used as the last modify time for the file. A value of zero indicates the last modified time should be unchanged.

smb_rsvd This six 16-bit field is only present if *smb_wct* is 12. These fields should be ignored.

smb_pad A single 8-bit field which is used to pad out the beginning of the *smb_data* area to a 32-bit address boundary.

smb_data A string of bytes, in no particular format, whose length is given by *smb_count*. This is the data to be written.

SMBwriteclose Error Code Descriptions

Exactly the errors returned by *SMBwriteX* and *SMBclose* can be returned for this request. If an error occurs during the write operation, the file will still be closed. Only one error can be returned in the response; if errors occur during both the write and close operations, the close error is reported.

SMBwriteclose Preconditions

1. The SMB redirector has sent a valid SMB with a TID for a writable resource.
2. The FID is valid and the process has write access to the file.

SMBwriteclose Postconditions

1. The data in the call is written to the file. If an error occurred, it will be reported unless a close error occurs as well.
2. The file is closed and any errors are reported.

SMBwriteclose Side Effects

Any buffered data for the file is written, and any outstanding locks are released in random order.

Conventions

- Locking (see Section 4.4 on page 33).

Chapter 11

Extended 1.0 SMB Connection Management Requests

This section defines those elements of the extended 1.0 SMB protocol dialects which support connection and LMX session management. They are:

SMBnegprot	negotiate modifications when an extended dialect is selected by the LMX server
SMBsecpkgX	negotiate security packages and related information
SMBsesssetupX	set up a session, log on a user
SMBtconX	extended Tree Connect

11.1 SMBnegprot Specification

SMBnegprot Detailed Description

This SMB protocol request is sent to establish the protocol dialect that the SMB redirector and LMX server will use when communicating with each other. The SMB redirector sends a list of dialects that it can use for communication. The LMX server responds with a selection of one of those dialects (numbered 0 to *n*) or −1 indicating that none of the dialects were acceptable. Exactly one negotiate message must be sent on each NetBIOS session; subsequent negotiate requests must be rejected with an error response and no action will be taken. The rules to the use of *SMBnegprot* outlined in Section 6.1 on page 55 hold here as well.

SMBnegprot Deviations

None.

SMBnegprot Field Descriptions

Field descriptions for other dialects of the SMB protocol (*SMBnegprot*) are:

From SMB redirector		To SMB redirector	
Field Name	Field Value	Field Name	Field Value
smb_com	*SMBnegprot*	*smb_com*	*SMBnegprot*
smb_wct	0	*smb_wct*	13
smb_bcc	min = 2	*smb_vwv*[0]	*smb_index*
smb_buf[]	dialect0	*smb_vwv*[1]	*smb_secmode*
	.	*smb_vwv*[2]	*smb_maxxmt*
	.	*smb_vwv*[3]	*smb_maxmux*
	dialect*n*	*smb_vwv*[4]	*smb_maxvcs*
		smb_vwv[5]	*smb_blkmode*
		smb_vwv[6,7]	*smb_sesskey*
		smb_vwv[8]	*smb_srv_time*
		smb_vwv[9]	*smb_srv_date*
		smb_vwv[10]	*smb_srv_tzone*
		smb_vwv[11-12]	*smb_rsvd*
		smb_bcc	
		smb_buf[]	*smb_cryptkey*[]

The fields are defined as:

dialect*n* A Dialect (type 02) buffer containing the name of a dialect (refer to Section 5.4 on page 48).

smb_index The dialect selected by the LMX server; corresponds to the *index*th dialect string in the request, where the first string is numbered 0.

smb_secmode This flag field describes the LMX server's security mode.

- Bit 0 If set, the LMX server is in user-level security mode; if clear, share-level.
- Bit 1 If set, the LMX server supports password encryption in SMB form (see Section 11.3 on page 144 and Appendix D on page 279).
- Bit 2 If set, the LMX server supports the *SMBsecpkgX* extended security package negotiation (see Section 11.2 on page 139).
- Bit 3-15 Reserved; MBZ (Must Be Zero).

smb_maxxmt The LMX server's maximum SMB buffer size in bytes. Minimum value is 1K byte. This provides sufficient room for most requests and responses. All SMB requests including chained requests must fit in this buffer size.

This is the maximum SMB message size which the SMB redirector can send to the LMX server. This size may be larger than the *smb_bufsize* value in the *SMBsesssetupX* request, sent to the LMX server from the SMB redirector, which is the maximum SMB message size the LMX server may send to the SMB redirector.

For example, if the LMX server's buffer size (*smb_maxxmt* in the *SMBnegprot* response) were 4K byte and the SMB redirectors's buffer size were only 2K byte (*smb_bufsize* in the *SMBsesssetupX* request), the SMB redirector could send up to 4K byte of data in an *SMBwrite* (or *SMBwriteX*) request but may request no more than 2K byte of data in *SMBread* (or *SMBreadX*) requests. The largest

response from the LMX server would also be 2K byte.

smb_maxmux The maximum number of simultaneous multiplexed reads supported per LMX session; must be at least 1.

smb_maxvcs The maximum number of NetBIOS sessions supported per LMX session. Must be 1.

smb_blkmode Whether or not *SMBreadbraw* and *SMBwritebraw* are supported.

Bit 0 If set, *SMBreadbraw* is supported.

Bit 1 If set, *SMBbwritebraw* is supported.

Bit 2-15 Reserved; Must Be Zero.

Some SMB redirectors when negotiating LANMAN 1.0 dialect ignore these bits and assume both SMBs are acceptable.

smb_sesskey A 32-bit value of the LMX session key; uniquely identifies an LMX session.

smb_srv_time 16-bit current time according to the LMX server (see Section 5.3.1 on page 43).

smb_srv_date 16-bit current date according to the LMX server (see Section 5.3.2 on page 43).

smb_srv_tzone A 16-bit value for the number of minutes the current time zone is away from GMT.

smb_rsvd A 32-bit reserved field. Must be zero.

smb_bcc In the case of *SMBnegprot*, the field gives the length of the token in *smb_cryptkey*.

smb_cryptkey This is an unformatted array of bytes which contains an opaque token to be used for password encryption (see Section 11.2 on page 139, Section 11.3 on page 144 and Appendix D on page 279).

Note that bit 0 of the *smb_flg* field in the SMB header of the response will be interpreted by the SMB redirector to indicate support for *SMBlockread* and *SMBwriteunlock*.

SMBnegprot Error Code Descriptions

If any error occurs, the LMX server will return <ERRSRV, ERRerror>; otherwise, <SUCCESS, SUCCESS> will be returned.

SMBnegprot Preconditions

The SMB redirector attempting to negotiate a protocol must have established a NetBIOS session with the LMX server.

SMBnegprot Postconditions

The SMB redirector that negotiated this protocol must be able to handle all aspects of the SMB dialect negotiated.

SMBnegprot Side Effects

The LMX server will keep record of which dialect the SMB redirector negotiated and will use only that dialect in conversations with the SMB redirector.

If the SMB redirector is to perform password encryption, it must store and use the *smb_cryptkey* token in accordance with the encryption function selected (see Section 11.2 on page 139) or with the SMB encryption mechanism (see Section 11.3 on page 144 and Appendix D on page 279).

Conventions

None.

11.2 SMBsecpkgX Specification

SMBsecpkgX Detailed Description

The *SMBsecpkgX* extended protocol request is used to negotiate the security package to be used for a given LMX session. Part of the negotiation determines the authentication and password encryption algorithms required to establish the identity of the user sitting at the SMB redirector system. The *SMBsecpkgX* request and response are only used when the LMX server is in user-level security mode and both the SMB redirector and the LMX server understand Extended User Authentication (see Section 2.2 on page 5).

The SMB redirector will send an *SMBsecpkgX* request to the LMX server immediately after receipt of an *SMBnegprot* response which set bits 1 and 2 in the *smb_secmode* field, only if the SMB redirector supports Extended User Authentication.

An LMX server may reject an *SMBsesssetupX* request which was not preceded by an acceptable *SMBsecpkgX* exchange, or it may instead support SMB-style authentication and encryption mechanisms (see Section 11.3 on page 144). An LMX server may provide a mechanism to control this choice, on either a per-server or per-share basis.

In addition to supporting negotiation of a security package and its components, the *SMBsecpkgX* exchange also supports a mechanism for authentication of the serving system to the SMB redirector similar to the SMB redirector to the LMX server mechanism supported by the combination of *SMBnegprot* and *SMBsesssetupX* requests.

After the successful exchange of *SMBsecpkgX* request and response the SMB redirector will use as its UID for the LMX session the value of the *smb_uid* field in the response header. This is the only case in which the LMX server selects the value of *smb_uid* to be used for the LMX session. In all other cases (that is, no *SMBsecpkgX* exchange) the value of *smb_uid* is selected by the SMB redirector.

SMBsecpkgX Deviations

Use of the *SMBsecpkgX* exchange is only defined for the client-server dialogue package-type. An LMX server may implement other package-types without conflict.

Within the client-server package-type negotiation, only the X/Open security package is defined. An LMX server may choose to support additional packages of that type.

SMBsecpkgX Field Descriptions

From SMB redirector		To SMB redirector	
Field Name	Field Value	Field Name	Field Value
smb_com	*SMBsecpkgX*	*smb_com*	*SMBsecpkgX*
smb_wct	4	*smb_wct*	4
smb_vwv[0]	*smb_com2*	*smb_vwv*[0]	*smb_com2*
smb_vwv[1]	*smb_off2*	*smb_vwv*[1]	*smb_off2*
smb_vwv[2]	*smb_pkgtype*	*smb_vwv*[2]	*smb_index*
smb_vwv[3]	*smb_numpkgs*	*smb_vwv*[3]	*smb_pkgarglen*
smb_bcc	min=4	*smb_bcc*	
smb_buf[]	*smb_pkglist* 1	*smb_buf*[]	*smb_pkgargs*
	.		
	.		
	smb_pkglist n		

smb_pkgtype A 16-bit field containing the package-type being negotiated by this *SMBsecpkgX* request and response. The only value defined is 0, the package-type for the dialogue between an SMB redirector and the LMX server.

smb_numpkgs A 16-bit integer containing the number of packages of type *smb_pkgtype* being offered by the SMB redirector. This must be greater than zero.

smb_pkglist Each *smb_pkglist* is a structure describing a particular package. The structures are concatenated together, with no padding, to form the *smb_buf* section of the request.

The *smb_pkglist* structure looks like:

Field Name	Field Type	Contents
smb_pkgnamlen	16-bit field	Length, in bytes, of package name in this structure.
smb_pkgarglen	16-bit field	Length of package-specific info (in bytes).
smb_pkgname	byte array	The name of the package described by this structure. This is not padded.
smb_pkgargs	byte array	Package-specific information. The format of this counted array is defined by the package name associated with it.

smb_index A 16-bit integer containing the number of the package selected by the LMX server. The first *smb_pkglist* in the request corresponds to an *smb_index* value of 0, the second corresponds to 1, etc. If the LMX server can support none of the offered packages, a –1 is returned.

smb_pkgarglen A 16-bit integer giving the length, in bytes, of the package-specific information being returned from the LMX server to the SMB redirector. This may be zero for some packages.

smb_pkgargs This is an unstructured array of bytes containing package-specific information in a format determined by the package selected by *smb_index*. The format may be different from that of the *smb_pkgargs* in the request for the same package.

X/Open has defined one package of type 0, this package has *smb_pkgname X/OPEN*. The *smb_pkgargs* for this package are defined below.

Request		Response	
Type	Name	Type	Name
16-bit field	*xp_flags*	16-bit field	*xp_esel*
string	*xp_name*	16-bit field	*xp_usel*
16-bit field	*xp_edialects*	type 01	*xp_ouinf*
string	*xp_e0*	type 01	*xp_nuinf*
...	...	type 01	*xp_Cr*
string	*xp_en*		
16-bit field	*xp_udialects*		
string	*xp_u0*		
...	...		
string	*xp_un*		
type 01	*xp_Cs*		

xp_flags A set of flags modifying use of this exchange.

Bit 0 If set, the LMX server must respond to the challenge, Cs, contained in this request. If clear, the SMB redirector does not require the LMX server to authenticate itself.

Bits 1-15 Undefined; MBZ (Must Be Zero).

xp_name A null-terminated string containing the username. This name, possibly truncated, should be used by the LMX server to identify which user is to be authenticated.

xp_edialects The number of bi-directional encryption function (referred to as E()) names which follow in the *pkgargs* structure. This must be greater than zero.

xp_en Each null-terminated string names a particular E() function. The meaning of these names must be agreed upon by implementors of SMB redirectors and LMX servers.

xp_udialects The number of password encryption function (U()) names which follow. This must be greater than zero.

xp_un Each null-terminated string names a particular U() function. As with E() functions, the meaning of these names must be mutually agreed upon by SMB redirector and LMX server systems.

xp_Cs This data (type 01) buffer contains a challenge string. The response string, *xp_Cr*, will be generated using the E() selected by the LMX server, and the password stored on the LMX server for the user indicated by *xp_username*. The SMB redirector can use the password, as typed by the user, *xp_ouinf*, and the challenge response to ensure that the LMX server in fact knew the user's password as well. The particular algorithm for accomplishing this depends upon the E() and U() functions negotiated. This field is meaningless and should be ignored if bit 0 of *xp_flags* is not set.

xp_esel The index of the *xp_en* which the LMX server has selected. This index is zero-based, in the same fashion as *smb_index* (above). If none of the offered *xp_en* functions are supported by the LMX server, a –1 will be returned in this field and an error will be returned.

xp_usel The index of the *xp_un* which the LMX server has selected. This index is zero-based. If none of the offered *xp_un* functions are supported by the LMX server, a –1 will be returned in this field and an error will be returned.

xp_ouinf A data (type 01) buffer, whose contents are used in combination with the user's password and the chosen U() to reproduce the password as stored on the LMX server. This string may be unused for some U() and would be of zero length if such a U() were selected.

xp_nuinf A data buffer whose contents are to be used if the password for this user is changed via some administrative protocol. Some LMX servers may not support such an administrative protocol, and some U() functions require no such data or permit reuse of such data; in any of these cases, the length of this buffer will be zero.

xp_Cr A data buffer containing the response to *xp_Cs*; see above. This field will be ignored and should be of zero length if bit 0 of *xp_flags* was not set.

SMBsecpkgX Error Code Descriptions

CAE Code	DOS Class	DOS Code	Description
-	ERRSRV	ERRbadpermits	For either the E() or U() functions, there was no match between the functions supported on the SMB redirector and LMX server.
-	ERRSRV	ERRerror	The SMB redirector has already negotiated this package-type.
-	ERRSRV	ERRbaduid	The UID given (*smb_uid*) is not known as a valid ID on this LMX session.
-	SUCCESS	SUCCESS	Everything worked, no problems.

If the user named in the *xp_name* field does not exist on the LMX server, the LMX server should nonetheless generate a properly formatted response with data that appears to be valid. The SMB redirector attempt to set up an LMX session should be rejected after the *SMBsesssetupX* request is received.

SMBsecpkgX Preconditions

The LMX server must have set bits 1 and 2 of the *smb_secmode* field in its *SMBnegprot* response on this same NetBIOS session.

SMBsecpkgX Postconditions

If the optional SMB redirector challenge was used, the SMB redirector can rely upon the LMX server actually knowing the user's password.

SMBsecpkgX Side Effects

All authentication exchanges after this SMB exchange will use the selected E() as an encryption and decryption mechanism. All passwords passed over the connection after this SMB exchange will be encoded using the selected U() and *xp_ouinf*/*xp_nuinf* information.

Conventions

- Chaining (see Section 3.9 on page 22).

Only *SMBsesssetupX* may be chained to *SMBsecpkgX*. Furthermore, this can only be successfully done if:

1. Only one E() and U() function is offered in the *SMBsecpkgX* request. If distinct functions are offered, the SMB redirector cannot know *a priori* which E() or U() function to use to compute the encrypted user password.
2. The U() function does not require the use of *xp_ouinf* to compute the encrypted password.

11.3 SMBsesssetupX Specification

SMBsesssetupX Detailed Description

This extended protocol request is used to further set up the LMX session normally just established via the *SMBnegprot* request/response. The *SMBsesssetupX* request serves two purposes: identification of the user for this LMX session, and negotiation of SMB redirector-side communication parameters.

- User Identification

  The actual semantics for this request are governed by the security mode of the LMX server. See Section 2.2 on page 5 for a discussion of these modes.

  In user-level security mode, the SMB redirector will establish a mapping between a particular username on the LMX server and a UID which the SMB redirector will use to represent that user. A password may be sent by the SMB redirector to authenticate that the person using the SMB redirector is indeed the username to be mapped to. Further, the password may be encrypted to ensure security.

  The LMX server validates the username and password supplied and, if valid, it establishes a mapping between the LMX session's UID and the actual UID corresponding to the specified username and password. That actual UID will be used for access checks required by requests issued on behalf of the UID on this LMX session.

  The value of the UID is relative to an LMX session; it is possible for the same UID value to represent two different users on two different LMX sessions on the LMX server. The LMX server must map the pair of <LMX session ID, UID> to the different accounts.

  In share-level security mode, the username and password are unused. The LMX server should use a unique, reserved account and corresponding actual UID to perform access checks for all requests.

- SMB Redirector Communications Parameters

  The LMX server, in its response to the *SMBnegprot* request, has set some parameters for the communication it was expecting from the SMB redirector. In the *SMBsesssetupX* request, the SMB redirector must indicate the parameters for the communication it is expecting from the LMX server. These values may be different; for example, the LMX server may be able to receive a maximum message size of 16K bytes, while the SMB redirector can only receive 1K bytes.

  Some LMX servers may need to renegotiate buffer sizes after the *SMBsesssetupX* exchange. This renegotiation is available through the *SMBtcon* request, but not through *SMBtconX*.

SMBsesssetupX Deviations

None.

SMBsesssetupX Field Descriptions

From SMB redirector		To SMB redirector	
Field Name	Field Value	Field Name	Field Value
smb_com	*SMBsesssetupX*	*smb_com*	*SMBsesssetupX*
smb_wct	10	*smb_wct*	3
smb_vwv[0]	*smb_com2*	*smb_vwv*[0]	*smb_com2*
smb_vwv[1]	*smb_off2*	*smb_vwv*[1]	*smb_off2*
smb_vwv[2]	*smb_bufsize*	*smb_vwv*[2]	*smb_action*
smb_vwv[3]	*smb_mpxmax*	*smb_bcc*	0
smb_vwv[4]	*smb_vc_num*		
smb_vwv[5,6]	*smb_sesskey*		
smb_vwv[7]	*smb_apasslen*		
smb_vwv[8,9]	*smb_rsvd*		
smb_bcc	min val=0		
smb_buf[]	*smb_apasswd*		
	smb_aname		

smb_com2 and *smb_off2* descriptions can be found in Section 3.9 on page 22.

smb_bufsize The size of the largest message the SMB redirector is willing to receive. It must be true that *smb_bufsize* ≤ *smb_maxxmt* (see Section 6.1 on page 55).

smb_mpxmax The maximum number of requests which the SMB redirector will have outstanding on a single LMX session. It must be true that *smb_mpxmax* ≤ *smb_maxmux* (see Section 6.1 on page 55).

smb_vc_num Permits multiple LMX sessions to be associated with a single NetBIOS session. If zero (0), this LMX session is the first or only NetBIOS session. If *smb_vc_num* is zero (0) and there are other previously established LMX sessions still connected from this SMB redirector, it is recommended that the LMX server abort the previous LMX session to free up the resources held.

smb_sesskey A 32-bit integer which identifies to which LMX session that this NetBIOS session is associated. Ignored when *smb_vc_num* is zero (0). This value would be obtained from the *smb_sesskey* field in the response to the *SMBnegprot* associated with the LMX session this NetBIOS session is to be made a part of.

smb_apasslen Length of the *smb_apasswd* field.

smb_rsvd A 32-bit reserved field; the LMX server should ignore this field.

smb_apasswd A character string containing the password, possibly encrypted. Ignored by an LMX server in share-level security mode.

smb_aname An ASCIIZ (not type 04) buffer containing the username to be associated with *smb_uid* and validated with *smb_apasswd*. Ignored by an LMX server in share-level security mode. The length of this field is derived from the difference between *smb_bcc* and *smb_apasslen*.

smb_action A bit-encoded field indicating the results of a successful LMX session setup. If bit 0 is clear, everything went normally. If bit 0 is set, the LMX session was setup but a default or guest account was used instead of the account requested. (An LMX server in share-level security mode would set this bit).

SMBsesssetupX Error Code Descriptions

CAE Code	DOS Class	DOS Code	Description
-	ERRSRV	ERRerror	Internal LMX server error.
-	ERRSRV	ERRbadpw	Username and password pair was invalid.
-	ERRSRV	ERRtoomanyuids	LMX server does not support this many UIDs in one LMX session.
-	ERRSRV	ERRerror	No *SMBnegprot* request has been issued on this NetBIOS session.
-	ERRSRV	ERRnosupport	This request cannot be chained to the request which precedes it in this message.
-	SUCCESS	SUCCESS	Everything worked, no problems.

SMBsesssetupX Preconditions

1. The SMB redirector attempting the *SMBsesssetupX* must have established an LMX session with the LMX server and negotiated an extended protocol dialect.
2. The username and password must both be valid instances of those types.
3. *smb_com2* must be a legal chained command.
4. There are many other preconditions based upon the SMBs that may be chained. These are enumerated in the specifications for those SMBs.

SMBsesssetupX Postconditions

1. If there are no errors the value in *smb_uid* is used as a valid UID in future SMBs.
2. There are many other postconditions based upon the SMBs that may be chained. These are enumerated in the specifications for these SMBs.

SMBsesssetupX Side Effects

Conversion of paths to a canonical pathname is controlled by bit 4 of the *smb_flg* in the header of this request (see Section 5.1 on page 37).

Conventions

- Opportunistic Locking (see Section 3.8.2 on page 20).
- Chaining (see Section 3.9 on page 22).

The SMBs which may be chained after *SMBsesssetupX* are:

SMBchkpath	*SMBfunique*	*SMBopen*	*SMBsearch*	*SMBtconX*
SMBcopy	*SMBgetatr*	*SMBopenX*	*SMBsetatr*	*SMBunlink*
SMBcreate	*SMBmkdir*	*SMBrename*	*SMBsplopen*	*SMBtrans*
SMBdskattr	*SMBmknew*	*SMBrmdir*	*SMBsplretq*	NIL
SMBffirst	*SMBmv*			

11.4 SMBtconX Specification

SMBtconX Detailed Description

This extended protocol request will establish direct access to a resource (file system subtree, spooled device, etc.) on an LMX server. The functionality provided by this request matches very closely that of the core protocol *SMBtcon* request. The differences are:

1. *SMBtconX* permits another request to be chained to it (for example, *SMBopenX*).
2. A flag can be set in the *SMBtconX* request which will invalidate the TID in the request, then acquire a new TID for the requested resource and return it.
3. The maximum receive buffer sizes cannot be renegotiated.
4. The resource type need not be explicitly identified.

SMBtconX Deviations

None.

SMBtconX Field Descriptions

From SMB redirector		To SMB redirector	
Field Name	Field Value	Field Name	Field Value
smb_com	*SMBtconX*	*smb_com*	*SMBtconX*
smb_wct	4	*smb_wct*	2
smb_vwv[0]	*smb_com2*	*smb_vwv*[0]	*smb_com2*
smb_vwv[1]	*smb_off2*	*smb_vwv*[1]	*smb_off2*
smb_vwv[2]	*smb_flags*	*smb_bcc*	min val=3
smb_vwv[3]	*smb_spasslen*	*smb_buf*[]	*smb_service*
smb_bcc	min val=3		
smb_buf[]	*smb_spasswd*		
	smb_path		
	smb_dev		

smb_com2 and *smb_off2* descriptions can be found in Section 3.9 on page 22.

smb_flags A 16-bit field containing additional control flags. The only flag currently defined is bit 0; if set, the TID in the request is to be closed (as if an *SMBtdis* request were received for it) before the new resource is obtained.

smb_spasslen A 16-bit field giving the length of the *smb_spasswd* field. If this value is zero, *smb_bcc* must contain the end-of-string terminator (that is, a zero character) for the password value.

smb_spasswd A string of bytes containing the password for the resource. May be encrypted. Refer to Appendix D on page 279.

smb_path An ASCIIZ buffer (not type 04) containing the resource name preceded by the LMX servername (refer to Section 5.3.9 on page 46). For example, a resource called src residing on a server called lmserver1 would be referenced by \\lmserver1\src. If not specified by the SMB redirector, a zero byte must be present.

smb_dev An ASCIIZ buffer giving the resource type the SMB redirector will use to refer to the newly-attached resource. If this value is not of a well-known form to the LMX server it is treated as a wildcard; in this case, the LMX server will return the actual resource type (see Section 5.36 on page 45). in the *smb_service* field of the response. If not specified by the SMB redirector, a zero byte must be present.

smb_service An ASCIIZ buffer identifying the actual resource type corresponding to the requested resource.

SMBtconX Error Code Descriptions

CAE Code	DOS Class	DOS Code	Description
-	ERRSRV	ERRerror	Ran out of TIDs.
-	ERRSRV	ERRerror	First command on the NetBIOS session was not an *SMBnegprot*.
-	ERRSRV	ERRerror	LMX server internal error.
-	ERRSRV	ERRbadpw	Bad password; name/password pair in the *SMBtconX* is invalid.
-	ERRSRV	ERRinvnetname	Invalid resource name supplied in the *SMBtconX*.
-	SUCCESS	SUCCESS	Everything worked, no problems.

SMBtconX Preconditions

1. The SMB redirector attempting to setup this *SMBtconX* must have established an LMX session with the LMX server.
2. The *smb_path*, *smb_spasswd* and *smb_dev* must all be valid instances of those types.
3. The process attempting to setup this *SMBtconX* must have negotiated an extended protocol dialect (for example, LANMAN 1.0 or LM1.2X002).

SMBtconX Postconditions

1. If there are no errors the TID and service string are valid and may be used in future SMB requests.
2. If bit 0 in *smb_flags* was set, the resource defined by the TID in the request has been disconnected from this LMX session.

SMBtconX Side Effects

None.

Conventions

- Filename (see Section 3.5 on page 15).
- Chaining (see Section 3.9 on page 22).

Requests which may be chained to *SMBtconX* are:

SMBchkpath	*SMBfunique*	*SMBmv*	*SMBrmdir*	*SMBsplretq*
SMBcopy	*SMBgetatr*	*SMBopen*	*SMBsearch*	*SMBtrans*
SMBcreate	*SMBmkdir*	*SMBopenX*	*SMBsetatr*	*SMBunlink*
SMBdskattr	*SMBmknew*	*SMBrename*	*SMBsplopen*	NIL
SMBffirst				

Chapter 12

Extended 1.0 SMB File Operations

This section defines the elements of the extended 1.0 SMB protocol which provide for normal operations on files. They are:

SMBopenX	open of a file with chaining
SMBlockingX	locking on a file with chaining
SMBreadX	read from a file with chaining
SMBwritebraw	write block raw to a file
SMBwriteclose	write to a file and close it
SMBwriteX	write to a file with chaining

12.1 SMBopenX Specification

SMBopenX Detailed Description

This extended protocol request opens a file, providing enhanced functionality over that of *SMBopen*.

SMBopenX Deviations

1. The archive, system and hidden file attribute bits are treated according to the file attributes convention. Refer to Section 4.3.1 on page 30.
2. LMX servers which cannot maintain a creation time for their files will ignore the create time field.

SMBopenX Field Descriptions

From SMB redirector		To SMB redirector	
Field Name	Field Value	Field Name	Field Value
smb_com	*SMBopenX*	*smb_com*	*SMBopenX*
smb_wct	15	*smb_wct*	15
smb_vwv[0]	*smb_com2*	*smb_vwv*[0]	*smb_com2*
smb_vwv[1]	*smb_off2*	*smb_vwv*[1]	*smb_off2*
smb_vwv[2]	*smb_flags*	*smb_vwv*[2]	*smb_fid*
smb_vwv[3]	*smb_mode*	*smb_vwv*[3]	*smb_attributes*
smb_vwv[4]	*smb_sattr*	*smb_vwv*[4-5]	*smb_time*
smb_vwv[5]	*smb_attr*	*smb_vwv*[6-7]	*smb_size*
smb_vwv[6-7]	*smb_time*	*smb_vwv*[8]	*smb_access*
smb_vwv[8]	*smb_ofun*	*smb_vwv*[9]	*smb_type*
smb_vwv[9-10]	*smb_size*	*smb_vwv*[10]	*smb_state*
smb_vwv[11-12]	*smb_timeout*	*smb_vwv*[11]	*smb_action*
smb_vwv[13-14]	*smb_resv*	*smb_vwv*[12-13]	*smb_fileid*
smb_bcc	min=1	*smb_vwv*[14]	*smb_resv*
smb_buf[]	*smb_pathname*	*smb_bcc*	0

smb_com2 and *smb_off2* descriptions can be found in Section 3.9 on page 22.

smb_flags Controls various special actions. If bit 0 is set, the additional information (*smb_vwv*[3..10]) fields will be valid in the response. Bits 1 and 2 control opportunistic locking (see Section 3.8.2 on page 20). The other bits are reserved.

smb_mode The open mode for the file (see Section 5.3.5 on page 44).

smb_sattr The set of attributes that the file must have in order to be found while searching to see if it exists. Regardless of the contents of this field, normal files always match (see Section 5.3.3 on page 43).

smb_attr The set of attributes that the new file is to have if the file needs to be created (see Section 5.3.3 on page 43).

smb_time In the request, this is the 32-bit integer time to be assigned to the file as a time of creation (if the file must be created). In the response, this is the 32-bit integer time of last modification. Refer to Section 5.3.1 on page 43.

smb_ofun This open function field controls actions to be taken on the file during the open (see Section 5.3.8 on page 46).

smb_size In the request, this 32-bit integer is the number of bytes to be reserved on file creation or truncation. In the response, the 32-bit integer contains the number of bytes in the file after any open actions have been taken (see *smb_ofun* above). This field is advisory.

smb_timeout This 32-bit integer is the number of milliseconds to wait on a blocked open before returning without obtaining a resource. A value of zero (0) means no delay (that is, do not queue the request). A value of –1 indicates to wait forever. See Section 3.11 on page 25.

smb_pathname An ASCIIZ buffer containing the name of the file to be opened.

smb_fid An FID representing this open instance of the file.

smb_attributes A file attribute field describing the actual attributes of the file after the open. See Section 5.3.3 on page 43.

smb_access The actual access rights granted to this process (see Section 5.3.7 on page 46).

smb_type A resource type field (see Section 5.3.6 on page 45).

smb_state Describes the status of a named pipe as follows. Refer to the X/Open CAE Specification, IPC Mechanisms for SMB.

- Bit 15 Blocking. Zero (0) indicates that reads/writes block if no data is available; 1 indicates that reads/writes return immediately if no data is available.
- Bit 14 Endpoint. Zero (0) indicates SMB redirector end of a named pipe; 1 indicates the LMX server end of a named pipe.
- Bits 10-11 Type of named pipe. 00 indicates the named pipe is a stream mode pipe; 01 indicates the named pipe is a message mode pipe.
- Bits 8-9 Read Mode. 00 indicates to read the named pipe as a stream mode named pipe; 01 indicates to read the named pipe as a message mode named pipe.

smb_action Describes the results of the open operation. This 16-bit field contains two fields:

Bit 15 Lock Status. Set true only if an opportunistic lock was requested by the SMB redirector and was granted by the LMX server. This bit should be false (0) if no lock was requested, the lock could not be granted, or the LMX server does not support opportunistic locking.

Bits 0-1 Open Action. The LMX server should set this to match the requested action from the *smb_ofun* field:

1 The file existed and was opened.

2 The file did not exist and was therefore created.

3 The file existed and was truncated.

smb_fileid This 16-bit field is reserved; MBZ (Must Be Zero).

smb_resv Reserved; MBZ.

SMBopenX Error Code Descriptions

CAE Code	DOS Class	DOS Code	Description
EACCES	ERRDOS	ERRnoaccess	Component of path-prefix denies search permission.
EACCES	ERRDOS	ERRnoaccess	Access permission is denied for the named file.
EAGAIN	ERRDOS	ERRshare	File exists, mandatory file/record locking is set, and there are outstanding record locks on the file.
EEXIST	ERRSRV	ERRerror	The create could not occur due to the existence of a file that did not have matching attributes (*smb_sattr*).
EFAULT	ERRSRV	ERRerror	Path points outside the allocated address space of the process.
EINTR	ERRSRV	ERRerror	A signal was caught during some system call.
EISDIR	ERRDOS	ERRnoaccess	Named file is a directory and access is write or read/write.
EMFILE	ERRSRV	ERRerror	Maximum number of file descriptors are currently open in this process.
ENFILE	ERRDOS	ERRnofids	System file table is full.
ENOENT	ERRDOS	ERRbadfile	File does not exist, or component of pathname does not exist.
ENOSPC	ERRSRV	ERRerror	File must be created, and the system is out of resources necessary to create files.
ENOTDIR	ERRDOS	ERRbadpath	Component of path-prefix is not a directory.
ENXIO	ERRSRV	ERRerror	The requested file is a CAE special file and the system cannot support access to the file at this time.
EROFS	ERRSRV	ERRerror	File resides on read-only file system and requested access permission is write or read/write.
ETXTBSY	ERRSRV	ERRerror	File is pure procedure file that is being executed and requested access specifies write or read/write.
-	ERRSRV	ERRinvnid	Invalid TID.
-	ERRSRV	ERRinvdevice	Invalid resource type; TID does not refer to a printer share.
-	ERRSRV	ERRbaduid	The UID given (*smb_uid*) is not known as a valid ID on this LMX session.
-	SUCCESS	SUCCESS	Everything worked, no problems.

SMBopenX Preconditions

The SMB redirector has sent a valid SMB request with a valid TID which is at least writable by this process.

SMBopenX Postconditions

The named file was possibly created or truncated, and then opened.

SMBopenX Side Effects

If an opportunistic lock was granted, the notification mechanisms described in Section 3.8.2 on page 20 are active.

Conventions

- Access (see Section 4.3.2 on page 30).
- Attributes (see Section 4.3.1 on page 30).
- Filenames (see Section 3.5 on page 15).
- Opportunistic Locking (see Section 3.8.2 on page 20).
- Chaining (see Section 3.9 on page 22).

The following are the only valid chained requests for this SMB: *SMBread*, *SMBreadX*, *SMBioctl* and *NIL*.

12.2 SMBlockingX Specification

SMBlockingX Detailed Description

This extended protocol request is used to lock and/or unlock one or more byte ranges of a particular regular file.

If the number of unlock ranges is non-zero, the byte ranges indicated by byte offset and length will be unlocked.

If the number of lock ranges is non-zero, the byte ranges indicated by byte offset and length will be locked, if possible. Locking byte ranges beyond the EOF is permitted. Access is permitted to any SMB redirector using the file descriptor provided with the lock request, but only requests using the PID that did the locking may do the unlocking. Attempts to lock bytes that have been previously locked will fail.

If the LMX server is unable to acquire all of the locks that the SMB redirector requested (after waiting for the length of the timeout, if specified), all the locks acquired with this request will be removed and the entire request fails.

Closing a file with locks still in force causes the locks to be released in an undefined order.

SMBlockingX Deviations

LMX servers may choose not to support lock timeouts, and may treat all requests as though a timeout of 0 had been requested.

LMX servers may choose not to support read-only locks, and will treat any request for such a lock as though a read/write lock had been requested.

Locking requests generated within the SMB protocol have a 32-bit unsigned offset for the beginning of the lock. The mapping of this offset within the CAE system on behalf of the SMB redirector is implementation-dependent.

SMBlockingX Field Descriptions

From SMB redirector		To SMB redirector	
Field Name	Field Value	Field Name	Field Value
smb_com	*SMBlockingX*	*smb_com*	*SMBlockingX*
smb_wct	8	*smb_wct*	2
smb_vwv[0]	*smb_com2*	*smb_vwv*[0]	*smb_com2*
smb_vwv[1]	*smb_off2*	*smb_vwv*[1]	*smb_off2*
smb_vwv[2]	*smb_fid*	*smb_bcc*	0
smb_vwv[3]	*smb_locktype*		
smb_vwv[4-5]	*smb_timeout*		
smb_vwv[6]	*smb_unlocknum*		
smb_vwv[7]	*smb_locknum*		
smb_bcc	10*(number of lock/unlock structs)		
smb_buf[]	*smb_unlkrng* *smb_lkrng*		

smb_com2 and *smb_off2* descriptions can be found in Section 3.9 on page 22

smb_fid The FID to use to perform locks or unlocks.

smb_locktype A bit-field which specifies the type of locks (mode) to be placed on the file. The mode is ignored for performing unlocks. The bits are defined as follows:

Bit 0 If set, indicates read-only lock requested. If a read-only lock is granted, other read-only lock requests on the same range of bytes will be permitted, but read/write locks (bit 0 not set) will be denied until all the read-only locks are released. Support for this request is optional.

Bit 1 If set, this indicates that an opportunistic lock is being broken, and in the response thereto, this bit will be set by the LMX server in an *SMBlockingX* request sent to the SMB redirector under the conditions outlined in Section 3.8.2 on page 20.

Bits 2-15 Reserved; ignored by the LMX server on receipt of request, and set to zero by the LMX server when sending a request.

smb_timeout A 32-bit integer indicating the amount of time, in milliseconds, to wait in an attempt to acquire all requested locks. A value of zero signals the LMX server not to wait at all but to return an error immediately if any lock could be obtained. A value of –1 indicates the LMX server should wait indefinitely to obtain the locks. (Note that requests with –1 timeouts could easily lead to deadlock.) Support for this field is optional; an LMX server may ignore all values and behave as if a timeout of 0 (that is, no wait) was always requested (reference X/Open CAE Specification, IPC Mechanisms for SMB).

smb_unlocknum A signed 16-bit field indicating the number of *smb_unlkrng* structures attached to this request.

smb_locknum A signed 16-bit field indicating the number of *smb_lkrng* structures attached to this request.

The *smb_unlkrng* and *smb_lkrng* structures are identical. Each describes a range of bytes to be unlocked or locked, respectively. The structure is:

Position	Field Name	Description
00	*smb_lpid*	The PID of the process owning the lock.
02	*smb_lkoff*	A 32-bit unsigned integer containing the offset, in bytes, to the start of the range to be unlocked or locked.
06	*smb_lklen*	A 32-bit unsigned integer containing the length, in bytes, of the range to be unlocked or locked.

SMBlockingX Error Code Descriptions

See Section 7.7 on page 81 and Section 7.8 on page 83 for other error codes.

CAE Code	DOS Class	DOS Code	Description
-	ERRDOS	ERRbadfile	File was not found.
-	ERRDOS	ERRbadfid	An invalid FID was specified.
-	ERRDOS	ERRlock	A lock request conflicted with an existing lock, the mode specified was invalid, or an unlock request was attempted by other than the owning PID.
-	ERRSRV	ERRerror	Invalid SMB request was sent.
-	ERRSRV	ERRinvdevice	Requested a lock on a non-file system subtree.
-	ERRSRV	ERRinvnid	Invalid TID was specified.
-	ERRSRV	ERRbaduid	The UID given (*smb_uid*) is not known as a valid ID on this LMX session.
-	SUCCESS	SUCCESS	Everything worked, no problems.

SMBlockingX Preconditions

1. The SMB redirector has sent a valid SMB request.
2. The SMB redirector must have a valid TID to a file system subtree.
3. The SMB redirector has specified a valid FID and has appropriate privileges.

If the request is generated by the LMX server, the FID corresponds to a file which the SMB redirector had opened with an opportunistic lock.

SMBlockingX Postconditions

1. Locking a range of bytes will fail if any subranges or overlapping ranges are locked. In other words, if any of the specified bytes are already locked, the lock will fail.
2. Either all of the requested ranges will be locked or none will. That is, if a lock on any of the specified ranges fails, any of the ranges previously locked by this request will be unlocked. Locked ranges not locked by this request remain locked.
3. If the lock request timed out, the response will return an ERRlock as if a lock could not be obtained and a zero timeout was specified.

If the request was generated by the LMX server, any data being cached on the SMB redirector has been flushed and/or invalidated, and the LMX server can permit the operation which caused the opportunistic lock break to complete.

SMBlockingX Side Effects

Any process using the FID specified in the request has access to the locked bytes, but other processes will be denied the locking of the same bytes.

Conventions

- Access (see Section 4.3.2 on page 30).
- Attributes (see Section 4.3.1 on page 30).
- Locking (see Section 4.4 on page 33).
- Filenames (see Section 4.2 on page 28).
- Opportunistic Locking (see Section 3.8.2 on page 20).
- Chaining (see Section 3.9 on page 22).

The *SMBlockingX* request may only have an *SMBread* or *SMBreadX* chained request.

12.3 SMBreadX Specification

SMBreadX Detailed Description

The *SMBreadX* extended protocol request is used to read data from any of the supported file types mentioned in Section 3.7 on page 17. The request allows reads to be timed out and offers a generalised alternative to the *SMBread* request.

SMBreadX Deviations

Not all LMX servers support all types listed in Section 5.3.6 on page 45. Some LMX servers may ignore the *smb_timeout* and *smb_remaining* fields for some types.

SMBreadX Field Descriptions

From SMB redirector		To SMB redirector	
Field Name	Field Value	Field Name	Field Value
smb_com	*SMBreadX*	*smb_com*	*SMBreadX*
smb_wct	10	*smb_wct*	12
smb_vwv[0]	*smb_com2*	*smb_vwv*[0]	*smb_com2*
smb_vwv[1]	*smb_off2*	*smb_vwv*[1]	*smb_off2*
smb_vwv[2]	*smb_fid*	*smb_vwv*[2]	*smb_remaining*
smb_vwv[3-4]	*smb_offset*	*smb_vwv*[3-4]	*smb_rsvd*
smb_vwv[5]	*smb_maxcnt*	*smb_vwv*[5]	*smb_dsize*
smb_vwv[6]	*smb_mincnt*	*smb_vwv*[6]	*smb_doff*
smb_vwv[7-8]	*smb_timeout*	*smb_vwv*[7-10]	*smb_rsvd*
smb_vwv[9]	*smb_countleft*	*smb_bcc*	(data length + pad)
smb_bcc	0	*smb_buf*[]	*smb_pad*
			smb_data

smb_com2 and *smb_off2* descriptions can be found in Section 3.9 on page 22.

smb_fid The FID from which the data should be read.

smb_offset A 32-bit integer containing the offset into the file (in bytes) at which the read should start.

smb_maxcnt An unsigned 16-bit field indicating the maximum number of bytes to read. Note that a single *SMBreadX* request cannot return more than the minimum of *smb_maxcnt* and the maximum negotiated buffer size for the LMX session. (See Section 11.3 on page 144 and Section 6.1 on page 55).

smb_mincnt An unsigned 16-bit value indicating the minimum number of bytes to return.

smb_timeout A 32-bit integer containing the number of milliseconds the LMX server should wait before returning. If *smb_mincnt* bytes are read before this time has expired, the LMX server should generate a response immediately. For regular files this field is ignored.

When reading from a named pipe (refer to the X/Open Developers' Specification, Protocols for X/Open PC Interworking: SMB), there are several special values which the SMB redirector can specify in this field:

0 If no data is available in the named pipe, respond immediately with *smb_dsize* set to zero (0).

–1 Block forever until at least *smb_mincnt* bytes of data are available, and return that data.

–2 Use the default timeout associated with the named pipe being read (reference X/Open CAE Specification, IPC Mechanisms for SMB).

>0 Wait until *smb_mincnt* data bytes are available or the timeout occurs. If there is a timeout, respond with a timeout error and whatever data was available.

smb_countleft This unsigned 16-bit field contains a hint to the LMX server indicating approximately how many more bytes will be read from this FID before the next non-read operation is requested for it. This is generated to help the LMX server increase performance by reading ahead in the file in anticipation of another *SMBreadX* request. An LMX server may ignore this field.

smb_remaining This signed 16-bit integer is always –1 for regular files. For named pipes and CAE special files, this 16-bit integer indicates the number of bytes that could be read from this file without blocking. This value need only be an approximation, and it may become inaccurate after the response is sent back to the SMB redirector. An LMX server may choose not to support this functionality and always return –1.

smb_dsize This unsigned 16-bit field contains the number of bytes of data actually read and returned in this response.

smb_doff This unsigned 16-bit field indicates the offset from the SMB header to the start of the returned data, in bytes. This permits variable-sized padding.

smb_rsvd These two 16-bit and four 16-bit fields are padding that force the *SMBreadX* response to be the same size as the *SMBwriteX* request. They must be zero.

smb_pad This field is between zero and three 8-bit fields in length, as governed by the *smb_doff* field. It may be used by an LMX server to pad the size of the *SMBreadX* response out to a 16-bit or 32-bit boundary which provides the best performance.

smb_data The actual data read from the file.

SMBreadX Error Code Descriptions

For more information pertaining to potential error codes generated by this SMB request see Section 7.4 on page 73 and Section 7.10 on page 87.

CAE Code	DOS Class	DOS Code	Description
-	ERRDOS	ERRnoaccess	Access denied. The requester's context does not permit the requested action or a read request is in conflict with an existing lock.
-	ERRDOS	ERRbadfid	Invalid FID. The SMB redirector has attempted to use an FID not recognised by the LMX server.
-	ERRDOS	ERRlock	Attempt to read bytes which were locked for write.
-	ERRDOS	ERRbadaccess	Invalid open mode for the attempted operation (for example, reading a write-only file).
-	ERRSRV	ERRerror	Error is returned to SMB redirectors for non-specific errors such as corrupt SMB requests.
-	ERRSRV	ERRinvnid	Error is returned to SMB redirectors attempting some action with an invalid TID.
-	ERRSRV	ERRtimeout	The requested named pipe operation timed out.
-	ERRSRV	ERRbaduid	The UID given (*smb_uid*) is not known as a valid ID on this LMX session.
-	SUCCESS	SUCCESS	Everything worked, no problems.

SMBreadX Preconditions

1. SMB request, UID and TID are valid and represent the appropriate access rights to perform the action.
2. The FID must be valid, and the SMB redirector must have appropriate permissions for the read operation.

SMBreadX Postconditions

1. The read data is returned.
2. The LMX server's current file pointer (see Section 7.6 on page 79) is advanced by the amount of data actually read.

SMBreadX Side Effects

None for normal files.

For named pipes or CAE special files, the data that was read is removed; a repeated read at the same offset will return new data.

Conventions

- Chaining (see Section 3.9 on page 22).

Only *SMBclose* request may be chained to the *SMBreadX* request.

12.4 SMBwritebraw Specification

SMBwritebraw Detailed Description

The write block raw message exchange provides a high-performance mechanism for transferring large amounts of data to be written to a file on the LMX server. Any supported file type, including spool files, may be written with this exchange.

The *SMBwritebraw* exchange behaves much like an *SMBwritebmpx* exchange, except that instead of additional data being sent in secondary requests, all the additional data is sent in a single raw message; that is, the first segment of data is sent in the primary request, and the remainder in a single message with no SMB header or *SMBwritebraw* subheader.

If all the data to be written fits in the primary request, a zero-length secondary request is still sent; even if the secondary request is zero-length, a secondary response must be generated when write-through mode was specified.

If the LMX server is busy or otherwise unable to support the raw write of the remaining data, the data sent with the primary request is still written (to stable store if write-through mode was set). If any other error occurs, the data is discarded. In either case, an appropriate error is returned in a secondary response. A primary response is only sent if the primary request was satisfied with no errors and the LMX server is prepared for a raw message.

SMBwritebraw Deviations

The *smb_timeout* and *smb_remaining* fields will not be supported with I/O devices.

SMBwritebraw Field Descriptions

Primary *SMBwritebraw* (extended other than core plus):

From SMB redirector		To SMB redirector	
Field Name	Field Value	Field Name	Field Value
smb_com	*SMBwritebraw*	*smb_com*	*SMBwritebraw*
smb_wct	12	*smb_wct*	1
smb_vwv[0]	*smb_fid*	*smb_vwv*[0]	*smb_remaining*
smb_vwv[1]	*smb_tcount*	*smb_bcc*	0
smb_vwv[2]	*smb_rsvd*		
smb_vwv[3 4]	*smb_offset*		
smb_vwv[5 6]	*smb_timeout*		
smb_vwv[7]	*smb_wmode*		
smb_vwv[8 9]	*smb_rsvd*		
smb_vwv[10]	*smb_dsize*		
smb_vwv[11]	*smb_doff*		
smb_bcc	min=0		
smb_buf[]	*smb_pad*		
	smb_data		

smb_fid The FID of the file to be written to.

smb_tcount An unsigned 16-bit field giving the total number of bytes that will be written to the file. This value must be correct in at least one of the requests in the exchange; in other requests, it may be an over-estimate.

smb_rsvd These fields are reserved and should be ignored by the LMX server.

smb_offset A 32-bit integer giving the position in the file at which the bytes in the request should be written.

smb_timeout A 32-bit integer giving the number of milliseconds the LMX server may block while trying to complete the write. This value is ignored for regular files. For I/O devices and named pipes (refer to X/Open CAE Specification, IPC Mechanisms for SMB), the LMX server will wait this much time to complete the write. If *smb_timeout* is –1, the LMX server will wait indefinitely; if it is –2, the server will wait the default amount of time for the file. An LMX server may choose to treat all timeouts as 0, that is, do not block.

smb_wmode A 16-bit flag field controlling the write mode. If bit 0 is set, write-through mode is requested; the LMX server will write all data atomically and acknowledge the write with the secondary response. If clear, write-behind is permitted; the LMX server need not write atomically and need not report completion. If bit 1 is set, the LMX server should fill in the *smb_remaining* field in the primary response.

smb_dsize The number of data bytes in this request.

smb_doff The offset in bytes from the beginning of the SMB header to *smb_data*.

smb_pad Between zero and three unused bytes; the SMB redirector may use these to pad out the *smb_data* area to a properly-aligned boundary.

smb_data The actual data to be written. This is a string of bytes in no particular format.

smb_remaining A 16-bit integer which is always –1 for regular files or if bit 1 of *smb_wmode* is not set. Otherwise, this is the number of bytes available to be read from the I/O device or named pipe specified by the FID. If the LMX server does not support this functionality, –1 should always be returned.

Secondary *SMBwritebraw*:

From SMB redirector		To SMB redirector	
Field Name	Field Value	Field Name	Field Value
raw data		*smb_com*	*SMBwritec*
		smb_wct	1
		smb_vwv[0]	*smb_count*
		smb_bcc	0

smb_count The total number of bytes written. If this is different from the smallest *smb_tcount* sent by the SMB redirector, some error occurred (for example, out of free space on the file system).

SMBwritebraw Error Code Descriptions

CAE Code	DOS Class	DOS Code	Description
-	ERRDOS	ERRbadfid	Invalid FID.
-	ERRDOS	ERRnoaccess	File opened in deny write mode, or write range overlaps a lock.
-	ERRDOS	ERRbadaccess	Invalid open mode for the attempted operation.
-	ERRSRV	ERRerror	Corrupt SMB.
-	ERRSRV	ERRinvnid	Invalid TID.
-	ERRSRV	ERRnoresource	The LMX server is temporarily out of a needed resource.
-	ERRSRV	ERRtimeout	Requested operation timed out.
-	ERRSRV	ERRuseMPX	Can't do raw mode at this time; use *SMBwritebmpx*.
-	ERRSRV	ERRuseSTD	Can't do raw mode at this time; use *SMBwrite* or *SMBwriteX*.
-	ERRSRV	ERRbaduid	The UID given (*smb_uid*) is not known as a valid ID on this LMX session.
-	SUCCESS	SUCCESS	Everything worked, no problems.

SMBwritebraw Preconditions

1. The primary SMB was valid and specified a valid TID for a writable resource.
2. The FID was valid, and the process had write access to the file.
3. Before sending the secondary message, the LMX server must have sent a primary response. The LMX server has been able to write the accompanying data to disk, allocated the needed memory for a buffer, and sent the response to the SMB redirector.

SMBwritebraw Postconditions

1. If write-through mode is set, a primary response or secondary response indicates the data in the primary response has been written to stable store (unless some error other than ERRuseSTD or ERRuseMPX was returned).
2. After a primary response is received, the LMX server is ready for a raw secondary message.

SMBwritebraw Side Effects

None.

Conventions

- Locking (see Section 4.4 on page 33).

12.5 SMBwriteclose Specification

SMBwriteclose Detailed Description

The write and close protocol request is used to first write the specified bytes and then close the file. Any supported file type, including spool files, may be specified in this request. This request behaves identically to an *SMBwrite* or *SMBwriteX* request followed by an *SMBclose* request. Any buffered data must be flushed to stable store or to the device before the response is sent.

Since the call is related to either the *SMBwrite* or *SMBwriteX* request, the length of the request may change; an SMB redirector may construct the request like *SMBwrite*, with six 16-bit fields in the variable word vector, or like *SMBwriteX*, with twelve 16-bit fields in the *smb_vwv*. The LMX server must be prepared to accept either form.

SMBwriteclose Deviations

See Section 7.5 on page 76 and Section 12.6 on page 168 for details.

SMBwriteclose Field Descriptions

From SMB redirector		To SMB redirector	
Field Name	Field Value	Field Name	Field Value
smb_com	*SMBwriteclose*	*smb_com*	*SMBwriteclose*
smb_wct	(6 or 12)	*smb_wct*	1
smb_vwv[0]	*smb_fid*	*smb_vwv*[0]	*smb_count*
smb_vwv[1]	*smb_count*	*smb_bcc*	0
smb_vwv[2-3]	*smb_offset*		
smb_vwv[4-5]	*smb_time*		
smb_vwv[6-11]	*smb_rsvd*		
smb_bcc	(1+ *smb_count*)		
smb_buf[]	*smb_pad*		
	smb_data		

smb_fid The FID to be closed.

smb_count In the request, the number of bytes of data to be written. In the response, the number of bytes that were actually written.

smb_offset A 32-bit offset into the file, in bytes, at which the data is to be written.

smb_time A 32-bit time value to be used as the last modify time for the file. A value of zero indicates the last modified time should be unchanged.

smb_rsvd This six 16-bit field is only present if *smb_wct* is 12. These fields should be ignored.

smb_pad A single 8-bit field which is used to pad out the beginning of the *smb_data* area to a 32-bit address boundary.

smb_data A string of bytes, in no particular format, whose length is given by *smb_count*. This is the data to be written.

SMBwriteclose Error Code Descriptions

Exactly the errors returned by *SMBwriteX* and *SMBclose* can be returned for this request. If an error occurs during the write operation, the file will still be closed. Only one error can be returned in the response; if errors occur during both the write and close operations, the close error is reported.

SMBwriteclose Preconditions

1. The SMB redirector has sent a valid SMB with a TID for a writable resource.
2. The FID is valid and the process has write access to the file.

SMBwriteclose Postconditions

1. The data in the call is written to the file. If an error occurred, it will be reported unless a close error occurs as well.
2. The file is closed and any errors are reported.

SMBwriteclose Side Effects

Any buffered data for the file is written, and any outstanding locks are released in random order.

Conventions

- Locking (see Section 4.4 on page 33).

12.6 SMBwriteX Specification

SMBwriteX Detailed Description

This extended protocol request is used to write to any supported file type (see Section 3.7 on page 17). The *SMBwriteX* command allows writes to be timed out and offers a generalised alternative to the *SMBwrite* and *SMBsplwr* requests.

Note that a zero-length write does not truncate the file as was true of the *SMBwrite* request; rather a zero-length write merely transfers zero bytes of information to the file. The *SMBwrite* request may be used to truncate the file.

SMBwriteX Deviations

Some LMX servers may limit support of extended features for CAE special files. For example, *smb_timeout* and/or *smb_remaining* may not be supported and locking versus non-blocking may be a configured parameter, etc.

Some CAE systems provide no way for a programme to block until the local file cache has actually flushed to the disk, but simply indicate that a flush has been scheduled and will complete soon. An LMX server should nonetheless take steps to maximise the probability that the data is truly on disk before the SMB redirector is notified.

SMBwriteX Field Descriptions

From SMB redirector		To SMB redirector	
Field Name	Field Value	Field Name	Field Value
smb_com	*SMBwriteX*	*smb_com*	*SMBwriteX*
smb_wct	12	*smb_wct*	6
smb_vwv[0]	*smb_com2*	*smb_vwv*[0]	*smb_com2*
smb_vwv[1]	*smb_off2*	*smb_vwv*[1]	*smb_off2*
smb_vwv[2]	*smb_fid*	*smb_vwv*[2]	*smb_count*
smb_vwv[3,4]	*smb_offset*	*smb_vwv*[3]	*smb_remaining*
smb_vwv[5,6]	*smb_timeout*	*smb_vwv*[4,5]	*smb_rsvd*
smb_vwv[7]	*smb_wmode*	*smb_bcc*	0
smb_vwv[8]	*smb_countleft*		
smb_vwv[9]	*smb_rsvd*		
smb_vwv[10]	*smb_dsize*		
smb_vwv[11]	*smb_doff*		
smb_bcc	min=0		
smb_buf[]	*smb_pad*		
	smb_data		

smb_fid The FID handle of the file to which the data should be written.

smb_offset A 32-bit unsigned integer giving the position in the file at which the data is to be written.

smb_timeout A 32-bit signed field giving the time (in milliseconds) within which a write must complete. A value of zero (0) indicates the write should never block. This field is ignored for regular files.

For other than regular file types (refer to X/Open CAE Specification, IPC Mechanisms for SMB), this value has two special values. If the timeout is –1, the LMX server should block indefinitely waiting for the write. If the timeout is –2, the LMX server should use the default timeout for the file type.

smb_wmode A 16-bit field containing flags, defined as follows:

- Bit 0 If set, an LMX server must not respond to the SMB redirector before the data is actually written to the disk (that is, write-through).
- Bit 1 If set, the LMX server should set *smb_remaining* correctly for writes to named pipes or I/O devices.
- Bit 2 For named pipes only. If set, *RawwriteNamedPipe* should be used. (See the X/Open CAE Specification, IPC Mechanisms for SMB).
- Bit 3 For named pipes only. If set, this data is the start of a message.

All other bits are reserved and should be ignored.

smb_countleft This unsigned 16-bit field is an advisory field telling the LMX server approximately how many bytes will be written to this file before the next non-write operation. It should include the number of bytes to be written by this request. An LMX server may ignore this field or use it to perform optimisations.

smb_rsvd A 16-bit reserved field; MBZ.

smb_dsize An unsigned 16-bit field giving the amount of data to be written, in bytes.

smb_doff A 16-bit field giving the offset from the start of the SMB header to the beginning of the data to be written. Specifying this field allows an SMB redirector to efficiently align the data buffer.

smb_pad The 8-bit fields between the end of the *SMBwriteX* header and the beginning of the data as pointed to by *smb_doff*. These fields should be ignored.

smb_data The actual data to be written. This is not in a buffer form; it is simply a string of bytes.

smb_count A 16-bit field giving the actual number of bytes written. The value would be different from *smb_dsize* if, for example, the file system became full or a file size limit imposed by *ulimit* was reached (refer to Section 4.3.3 on page 30).

smb_remaining This 16-bit integer should be –1 for regular files. For named pipes and I/O devices, if bit 1 of *smb_wmode* is set, the server should return the amount of data available to be read on this named pipe after the read. This value may be approximate, and a server may simply force this field to be –1.

smb_rsvd A 32-bit reserved field. It should be zero (0).

SMBwriteX Error Code Descriptions

CAE Code	DOS Class	DOS Code	Description
-	ERRDOS	ERRnoaccess	TID non-writable or other prohibition of access.
-	ERRDOS	ERRbadfid	Invalid FID. The SMB redirector has attempted to use an FID not recognised by the LMX server.
-	ERRDOS	ERRlock	The write overlapped an existing byte-range lock placed by another process.
-	ERRDOS	ERRbadaccess	Invalid open mode for the attempted operation (for example, writing a read-only file).
-	ERRSRV	ERRerror	Error is returned to the SMB redirector for non-specific errors such as corrupt SMB requests.
-	ERRSRV	ERRinvnid	Invalid TID.
-	ERRSRV	ERRtimeout	The requested operation timed out.
-	ERRSRV	ERRbaduid	The UID given (*smb_uid*) is not known as a valid ID on this LMX session.
-	SUCCESS	SUCCESS	Everything worked, no problems.

SMBwriteX Preconditions

SMB request, UID and TID are valid and represent the appropriate access rights to perform the action.

SMBwriteX Postconditions

If no error occurred, the data was buffered to be written to disk. The current file pointer for this file is advanced.

SMBwriteX Side Effects

A write-through write will cause the written data to be flushed to stable store, and may cause all buffered data for the file to be flushed.

Coonventions

Chaining (see Section 3.9 on page 22).

The following are the only valid requests which may be chained to an *SMBwriteX* request: *SMBread*, *SMBreadX*, *SMBlockingX*, *SMBclose*, *SMBlockread* and NIL.

12.7 SMBreadbmpx Specification

SMBreadbmpx Detailed Description

The read block multiplexed request is used to maximise the performance of reading a large block of data from the LMX server to the SMB redirector on a multiplexed LMX session. The *SMBreadbmpx* request can be applied to any supported file type.

Each *SMBreadbmpx* request will cause one or more associated responses to be sent from the LMX server. Each response contains as much of the remaining data to be read as will fit, and responses are generated until all the requested data has been transmitted. The LMX server can rely on the SMB redirector to maintain synchronisation; if the SMB redirector encounters a problem while it is receiving responses to an *SMBreadbmpx* request, it is responsible for discarding all those responses and will not notify the LMX server in any way. After solving the problem, the SMB redirector may reissue the request; the LMX server need not retain state concerning a completed *SMBreadbmpx* request. No acknowledgement of receipt from the SMB redirector is needed; the underlying transport is expected to ensure all responses arrive at the SMB redirector in the correct order.

Note that the request and all responses make up a single complete SMB exchange; thus, the TID, PID and UID are expected to remain constant. Also, the *SMBreadbmpx* exchange is supported on multiplexed NetBIOS sessions. What this means is that the SMB redirector may issue other SMB requests while the (multiple) *SMBreadbmpx* responses are being sent from the LMX server to the SMB redirector. Because of this, the response must contain the MID and PID of the original *SMBreadbmpx* request.

During an *SMBreadbmpx* exchange, the SMB redirector should not issue SMB requests which conflict with this; for example, the SMB redirector should not issue an *SMBclose* request on the same file for which it is still receiving *SMBreadbmpx* responses.

SMBreadbmpx Deviations

LMX servers may not support timeouts on all possible file types.

SMBreadbmpx Field Descriptions

From SMB redirector		To SMB redirector	
Field Name	Field Value	Field Name	Field Value
smb_com	*SMBreadbmpx*	*smb_com*	*SMBreadbmpx*
smb_wct	8	*smb_wct*	8
smb_vwv[0]	*smb_fid*	*smb_vwv*[0-1]	*smb_offset*
smb_vwv[1-2]	*smb_offset*	*smb_vwv*[2]	*smb_tcount*
smb_vwv[3]	*smb_maxcnt*	*smb_vwv*[3]	*smb_remaining*
smb_vwv[4]	*smb_mincnt*	*smb_vwv*[4-5]	*smb_rsvd*
smb_vwv[5-6]	*smb_timeout*	*smb_vwv*[6]	*smb_dsize*
smb_vwv[7]	*smb_rsvd*	*smb_vwv*[7]	*smb_doff*
smb_bcc	0	*smb_bcc*	min=0
		smb_buf[]	*smb_pad*
			smb_data

smb_fid The FID of the file to be read from.

smb_offset A 32-bit integer giving the position in the file at which to read (in the request) or the position in the file at which the data returned in this response began.

smb_maxcnt Maximum number of bytes to return; the desired read size.

smb_mincnt The minimum number of bytes to read. For regular files, this value is usually zero. When the timeout is used, this is the minimum number of bytes which will satisfy the read; if fewer bytes are available, the request will block until enough are available or the timeout is reached.

smb_timeout A 32-bit integer giving the number of milliseconds to wait for *smb_mincnt* bytes of data to become readable. A timeout of zero (0) indicates the call should never block. This value is ignored for regular files and may be ignored for I/O devices. For named pipes, there are two special values: –1 means the request should block forever until at least *smb_mincnt* bytes become available; –2 means the default timeout associated with the named pipe should be used.

smb_rsvd These fields are reserved and should be ignored in requests and set to zero in responses.

smb_tcount An integer giving the total number of bytes expected to be returned in all responses to this request. This value will usually start at *smb_maxcnt* and may be reduced by file truncations while the read is in progress, etc. This value must be accurate in at least the last response generated (that is, contain the actual number of bytes sent in all responses) but may be an overestimate in earlier responses.

If this value in the last response is less than *smb_maxcnt*, EOF was encountered during the read. If this value is exactly zero (0), the original offset into the file began after EOF; in this case, only one response may be generated.

smb_remaining This integer should be –1 for regular files. For devices or named pipes this indicates the number of bytes remaining to be read from the file *after* the bytes returned in the response were de-queued. LMX servers need not support this function and should return –1 if they do not support it.

smb_dsize The number of data bytes returned in the individual response.

smb_doff The offset in bytes from the beginning of the SMB to the beginning of the data being returned. This offset permits the LMX server to use an efficient alignment of the data within the SMB response.

smb_pad Zero (0) to three (3) bytes of padding. This is the space after the end of the *SMBreadbmpx* subheader which is unused because the data was aligned. The *smb_doff* points to the first byte *after* this bytestring.

smb_data The actual data bytes read.

SMBreadbmpx Error Code Descriptions

See Section 12.3 on page 160 for other error codes.

CAE Code	DOS Class	DOS Code	Description
-	ERRDOS	ERRnoaccess	File was opened in Deny Read mode.
EBADFID	ERRDOS	ERRbadfid	The FID was valid but unacceptable to the underlying OS.
-	ERRDOS	ERRlock	Read overlapped a byte-range lock granted to another process.
-	ERRDOS	ERRbadaccess	Some conflict in open mode occurred.
-	ERRSRV	ERRerror	Invalid SMB.
-	ERRSRV	ERRinvnid	Invalid TID.
-	ERRSRV	ERRnoresource	A temporary resource limitation in the LMX server caused this request to fail.
-	ERRSRV	ERRtimeout	A timeout occurred.
-	ERRSRV	ERRuseSTD	Temporarily out of sufficient buffers.
-	ERRSRV	ERRbaduid	The UID given (*smb_uid*) is not known as a valid ID on this LMX session.
-	SUCCESS	SUCCESS	Everything worked, no problems.

SMBreadbmpx Preconditions

1. SMB request, UID and TID are valid and represent the appropriate access rights to perform the action.
2. The FID is valid.

SMBreadbmpx Postconditions

1. For I/O devices or named pipes the returned data was consumed from the device.
2. After completion the current file position pointer will be right after the read data or at EOF.

SMBreadbmpx Side Effects

Because of the nature of the request, the operation may not be atomic on the LMX server; requests on the same file from other processes may change the results of this request.

Conventions

- Locking (see Section 4.4 on page 33).

12.8 SMBwritebmpx Specification

SMBwritebmpx Detailed Description

This extended protocol request provides a high performance mechanism for writing large amounts of data while other activity is being generated by the SMB redirector. The *SMBwritebmpx* operation can be performed on any supported file type.

Unlike most SMBs, there are two forms of both request and response: primary and secondary. The collection of all requests and responses related to a given primary *SMBwritebmpx* request is called an *SMBwritebmpx* exchange.

An *SMBwritebmpx* exchange begins when the SMB redirector sends a primary request. This request sets many of the parameters for the exchange and contains the first part of the data to be written. If an error occurred while handling this request, the LMX server sends a secondary response indicating the error and ends the exchange; otherwise, the LMX server sends a primary response indicating it is ready for more data. Then, if the amount of data to be written is greater than what could fit in the primary request, the SMB redirector sends secondary requests until all data has been sent. If the exchange was in write-through mode, the LMX server sends a secondary response; otherwise, the LMX server relies on the transport to ensure delivery of all requests and does not generate an additional reply.

If an error occurs after the primary response is sent, any secondary requests must be discarded. If write-through mode was requested, error information is returned to the SMB redirector in the secondary response. If not, the error is cached and returned in the response to the next request issued by the SMB redirector for that file.

Other requests may be issued on the same LMX session while the exchange is in progress. The TID, PID, UID and MID are expected to be identical in all requests and responses in a given *SMBwritebmpx* exchange.

If write-through mode is specified, the LMX server will collect all the data and write it to the disk atomically; otherwise, in write-behind mode, the LMX server need not make this guarantee.

SMBwritebmpx Deviations

Timeouts for I/O devices are implementation-dependent.

Some CAE systems provide no way for a programme to block until the local file cache has actually flushed to the disk, but simply indicate that a flush has been scheduled and will complete soon. An LMX server should nonetheless take steps to maximise the probability that the data is truly on disk before the SMB redirector is notified.

SMBwritebmpx Field Descriptions

Primary Request/Response

From SMB redirector		To SMB redirector	
Field Name	Field Value	Field Name	Field Value
smb_com	*SMBwritebmpx*	*smb_com*	*SMBwritebmpx*
smb_wct	12	*smb_wct*	1
smb_vwv[0]	*smb_fid*	*smb_vwv*[0]	*smb_remaining*
smb_vwv[1]	*smb_tcount*	*smb_bcc*	0
smb_vwv[2]	*smb_rsvd*		
smb_vwv[3,4]	*smb_offset*		
smb_vwv[5,6]	*smb_timeout*		
smb_vwv[7]	*smb_wmode*		
smb_vwv[8,9]	*smb_rsvd*		
smb_vwv[10]	*smb_dsize*		
smb_vwv[11]	*smb_doff*		
smb_bcc	min=0		
smb_buf[]	*smb_pad*		
	smb_data		

smb_fid The FID of the file to be written to.

smb_tcount An unsigned 16-bit field giving the total number of bytes that will be written to the file. This value must be correct in at least one of the requests in the exchange; in other requests, it may be an over-estimate.

smb_rsvd These fields are reserved and should be ignored by the LMX server.

smb_offset A 32-bit integer giving the position in the file at which the bytes in the request should be written.

smb_timeout A 32-bit integer giving the number of milliseconds the LMX server may block while trying to complete the write. This value is ignored for regular files. For I/O devices and named pipes (refer to the X/Open CAE Specification, IPC Mechanisms for SMB), the LMX server will wait this much time to complete the write. If *smb_timeout* is –1, the LMX server will wait indefinitely; if it is –2, the server will wait the default amount of time for the file. An LMX server may choose to treat all timeouts as 0, that is, do not block.

smb_wmode A 16-bit flag field controlling the write mode. If bit 0 is set, write-through mode is requested; the LMX server will write all data atomically and acknowledge the write with the secondary response. If clear, write-behind is permitted; the LMX server need not write atomically and need not report completion. If bit 1 is set, the LMX server should fill in the *smb_remaining* field in the primary response.

smb_dsize The number of data bytes in this request.

smb_doff The offset in bytes from the beginning of the SMB header to *smb_data*.

smb_pad Between zero and three unused bytes; the SMB redirector may use these to pad out the *smb_data* area to a properly-aligned boundary.

smb_data The actual data to be written. This is a string of bytes in no particular format.

smb_remaining A 16-bit integer which is always –1 for regular files or if bit 1 of *smb_wmode* is not set. Otherwise, this is the number of bytes available to be read from the I/O device or named pipe specified by the FID. If the LMX server does not support this functionality, –1 should always be returned.

Secondary Request/Response

From SMB redirector		To SMB redirector	
Field Name	Field Value	Field Name	Field Value
smb_com	*SMBwritebs*	*smb_com*	*SMBwritec*
smb_wct	8	*smb_wct*	1
smb_vwv[0]	*smb_fid*	*smb_vwv*[0]	*smb_count*
smb_vwv[1]	*smb_tcount*	*smb_bcc*	0
smb_vwv[2-3]	*smb_offset*		
smb_vwv[4-5]	*smb_rsvd*		
smb_vwv[6]	*smb_dsize*		
smb_vwv[7]	*smb_doff*		
smb_bcc	min=0		
smb_buf[]	*smb_pad*		
	smb_data		

smb_count The total number of bytes written. If this is different from the smallest *smb_tcount* sent by the SMB redirector, some error occurred (for example, out of free space on the file system).

All other fields are identical to the primary request.

SMBwritebmpx Error Code Descriptions

For other error codes see Section 12.6 on page 168. If a secondary response is not being generated by the LMX server, any error should be cached and returned in the response to the next request from the same process involving this FID.

CAE Code	DOS Class	DOS Code	Description
-	ERRSRV	ERRnoresource	Unable to allocate enough buffer space.
-	ERRSRV	ERRtimeout	Timeout occurred.
-	ERRSRV	ERRuseSTD	Some resource limitation prevents the LMX server from supporting *SMBwritebmpx* at this time; more limited write requests (*SMBwrite*, *SMBwriteX*) should be used instead.
-	ERRSRV	ERRbaduid	The UID given (*smb_uid*) is not known as a valid ID on this LMX session.
-	SUCCESS	SUCCESS	Everything worked, no problems.

SMBwritebmpx Preconditions

1. The SMB redirector has sent a valid SMB request with a valid TID for a writable resource.
2. The FID is valid and the process has write access.

SMBwritebmpx Postconditions

1. After the LMX server responds to the primary request to write-behind, the data in the primary write-behind request has been written.
2. After the secondary response, either an error was returned or all the data was written atomically.
3. After the last secondary request in a write-behind mode exchange is received, all the data is available to be read but might not yet be written to stable store.
4. If write-through mode was not specified, the LMX server has cached any errors to be sent as a response to the next request from this process related to this file.

SMBwritebmpx Side Effects

Because write-behind mode does not guarantee atomic write of all data, it is possible that this exchange is interfered with. It is possible, for example, that data from other processes could be interspersed with the data written by an exchange.

Coventions

None.

Chapter 13

Extended 1.0 SMB Directory and Attribute Operations

This section defines the elements of the extended SMB protocol that support directory and attribute access. They are:

SMBffirst	start/continue an extended wildcard directory lookup
SMBfclose	end an extended wildcard directory lookup
SMBfunique	perform a one-time extended wildcard directory lookup
SMBgetattrE	get extended file attributes
SMBsetattrE	set extended file attributes

13.1 SMBffirst Specification

SMBffirst Detailed Description

The *SMBffirst* extended protocol request behaves exactly like the *SMBsearch* core request, except the LMX server can expect the SMB redirector to terminate the search by issuing an *SMBfclose* request. Because of this expectation, the LMX server should not use heuristics to terminate the search, and should instead preserve all search state and resources until the *SMBfclose* request is received or the LMX session is closed.

As in the case of *SMBsearch*, there are two forms of the *SMBffirst* request: *FindFirst*, indicated by a null *smb_search_id*, and *FindNext*, which has a valid *smb_search_id* specified.

If a *FindFirst* request (an *SMBffirst* request whose *smb_search_id* is null) fails (no entries are found), the LMX server should respond with a failure and terminate the search. No *SMBfclose* request should be expected.

Otherwise, *SMBffirst* behaves in all respects like *SMBsearch*.

SMBffirst Deviations

See Section 8.3 on page 99.

SMBffirst Field Descriptions

See Section 8.3 on page 99.

SMBffirst Error Code Descriptions

See Section 8.3 on page 99.

SMBffirst Preconditions

1. SMB request, UID and TID are valid and represent the appropriate access rights to perform the action on a searchable disk resource.
2. The process has read/search permissions on all directories encountered.
3. For a *FindNext* request, the matching *FindFirst/FindNext* request must not have failed.

SMBffirst Postconditions

1. If the *FindFirst* fails, the search is terminated.
2. As long as *SMBffirst* requests continue to succeed, search state and resources are maintained; directories may remain open, etc.
3. After each *FindNext*, state information is updated in such a way as to ensure the search can continue without returning *dir_info* on the same file twice.

SMBffirst Side Effects

Various directories may remain open for reading during the lifetime of an active search. This may interfere with requests from other processes on involved directories.

Coovention

- Access (see Section 4.3.2 on page 30).
- Attributes (see Section 4.3.1 on page 30).
- Filename (see Section 3.5 on page 15).
- Wildcard (see Section 3.6 on page 17).

13.2 SMBfclose Specification

SMBfclose Detailed Description

The *SMBfclose* extended protocol request terminates an active search begun by *SMBffirst*.

SMBfclose Deviations

None.

SMBfclose Field Descriptions

The *SMBfclose* request and response are identical to the *SMBsearch* request and response (see Section 8.3 on page 99). The fields are interpreted differently:

smb_com This should be *SMBfclose* in both request and response.

smb_count This 16-bit integer should be ignored in the request and must be zero in the response.

smb_attr This attribute field should be ignored.

smb_pathname This ASCIIZ (type 04) buffer should be empty; that is, the buffer contains a single ASCII NULL character.

smb_search_id This variable block (type 05) buffer should be one of the *find_buf_search_id* structures returned in any response to the search being terminated. This buffer identifies the search which is to be terminated.

smb_data This variable block (type 05) should be zero length; that is, the length for the buffer should be zero (0), and no data bytes should be appended.

SMBfclose Error Code Descriptions

Same as for *SMBsearch* (see Section 8.3 on page 99).

SMBfclose Preconditions

1. SMB request, UID and TID are valid and represent the appropriate access rights to perform the action.
2. The search identified by *smb_search_id* must be active.

SMBfclose Postconditions

Any allocated resources for the identified search are released, and the search is no longer active.

SMBfclose Side Effects

None.

Conventions

None.

13.3 SMBfunique Specification

SMBfunique Detailed Description

The *SMBfunique* extended 1.0 protocol request behaves exactly like the *SMBsearch* core request, except the LMX server can terminate the search immediately after sending the response. The *SMBfunique* request, while it does support a wildcard *smb_pathname*, is designed to return information on only a few (possibly one) files. If more files match than can fit into the response, the LMX server can disregard them.

SMBfunique Deviations

See Section 8.3 on page 99.

SMBfunique Field Descriptions

See Section 8.3 on page 99. The LMX server should expect that *smb_search_id* will always be a zero-length variable block (type 05) buffer.

SMBfunique Error Code Descriptions

See Section 8.3 on page 99.

SMBfunique Preconditions

1. SMB request, UID and TID are valid and represent the appropriate access rights to perform the action.
2. The process has read/search permissions on all directories encountered.

SMBfunique Postconditions

No state or resources are maintained on the LMX server after the response is sent; the search is considered inactive.

SMBfunique Side Effects

Because *SMBfunique* is a one pass search, interaction with other requests due to directories remaining open for long periods of time should be greatly reduced; however, they may not be eliminated.

Conventions

- Access (see Section 4.3.2 on page 30).
- Attributes (see Section 4.3.1 on page 30).
- Filename (see Section 3.5 on page 15).
- Wildcard (see Section 3.6 on page 17).

13.4 SMBgetattrE Specification

SMBgetattrE Detailed Description

This extended 1.0 protocol request returns extended attribute information for a given open regular file.

SMBgetattrE Deviations

1. LMX servers which cannot maintain a creation date and time for their files will return the last modify date and time instead.
2. The attribute field is treated according to the Attribute convention.

SMBgetattrE Field Descriptions

From SMB redirector		To SMB redirector	
Field Name	Field Value	Field Name	Field Value
smb_com	*SMBgetattrE*	*smb_com*	*SMBgetattrE*
smb_wct	1	*smb_wct*	11
smb_vwv[0]	*smb_fid*	*smb_vwv*[0]	*smb_cdate*
smb_bcc	0	*smb_vwv*[1]	*smb_ctime*
		smb_vwv[2]	*smb_adate*
		smb_vwv[3]	*smb_atime*
		smb_vwv[4]	*smb_mdate*
		smb_vwv[5]	*smb_mtime*
		smb_vwv[6 7]	*smb_datasize*
		smb_vwv[8 9]	*smb_allocsize*
		smb_vwv[10]	*smb_attr*
		smb_bcc	0

smb_fid The FID for which extended attribute information should be returned.

smb_cdate A date field giving the creation date for the file. See Section 5.3.2 on page 43

smb_ctime A time field giving the creation time for the file. See Section 5.3.1 on page 43

smb_adate A date field giving the last access date for the file.

smb_atime A time field giving the last access time for the file.

smb_mdate A date field giving the last modify date for the file.

smb_mtime A time field giving the last modify time for the file.

smb_datasize A 32-bit integer giving the current size of the file (offset to EOF) in bytes.

smb_allocsize A 32-bit integer giving the amount of space allocated to the file. LMX servers on systems which do not support pre-allocation of space will set this field to the same value as *smb_datasize*.

smb_attr An attribute field giving the attributes of the file (see Section 3.7 on page 17).

SMBgetattrE Error Code Descriptions

CAE Code	DOS Class	DOS Code	Description
EBADF	ERRDOS	ERRbadfid	Invalid or no longer an acceptable FID.
EINTR	ERRSRV	ERRerror	A signal was caught during a system call.
-	ERRSRV	ERRinvnid	TID specified in command is invalid.
-	ERRSRV	ERRinvdevice	TID not for a disk resource.
-	SUCCESS	SUCCESS	Everything worked, no problems.

SMBgetattrE Preconditions

1. SMB request, UID and TID are valid and represent the appropriate access rights to perform the action.
2. The FID must be valid.

SMBgetattrE Postconditions

None.

SMBgetattrE Side Effects

None.

Conventions

- Attribute (see Section 4.3.1 on page 30).

13.5 SMBsetattrE Specification

SMBsetattrE Detailed Description

This extended 1.0 protocol request is used to set extended attribute information for an open regular file.

SMBsetattrE Deviations

LMX servers which cannot maintain a creation time for their files will ignore the create date and time fields.

SMBsetattrE Field Descriptions

From SMB redirector		To SMB redirector	
Field Name	Field Value	Field Name	Field Value
smb_com	*SMBsetattrE*	*smb_com*	*SMBsetattrE*
smb_wct	7	*smb_wct*	0
smb_vwv[0]	*smb_fid*	*smb_bcc*	0
smb_vwv[1]	*smb_cdate*		
smb_vwv[2]	*smb_ctime*		
smb_vwv[3]	*smb_adate*		
smb_vwv[4]	*smb_atime*		
smb_vwv[5]	*smb_mdate*		
smb_vwv[6]	*smb_mtime*		
smb_bcc	min=0		
	smb_rsvd		

smb_fid The FID whose extended attributes are to be changed.

smb_cdate A date field containing the creation date for the file. See Section 5.3.2 on page 43

smb_ctime A time field containing the creation time for the file. See Section 5.3.1 on page 43

smb_adate A date field containing the last access date for the file.

smb_atime A time field containing the last access time for the file.

smb_mdate A date field containing the last modify date for the file.

smb_mtime A time field containing the last modify time for the file.

smb_rsvd A reserved character string; LMX servers should ignore this field.

SMBsetattrE Error Code Descriptions

CAE Code	DOS Class	DOS Code	Description
EACCES	ERRSRV	ERRaccess	The UID does not have appropriate privilege and is not the owner of the file.
EBADF	ERRDOS	ERRbadfid	Invalid or no longer an acceptable FID.
EINTR	ERRSRV	ERRerror	A signal was caught during the operation.
EPERM	ERRSRV	ERRaccess	The UID does not have appropriate privilege and is not the owner of the file.
EROFS	ERRSRV	ERRaccess	File system is read-only.
-	ERRSRV	ERRinvnid	TID specified in command is invalid.
-	ERRSRV	ERRinvdevice	TID does not specify a disk resource.
-	SUCCESS	SUCCESS	Everything worked, no problems.

SMBsetattrE Preconditions

1. SMB request, UID and TID are valid and represent the appropriate access rights to perform the action.
2. The FID is valid.

SMBsetattrE Postconditions

A file time and date will remain unchanged if the corresponding date and time in the request was zero.

SMBsetattrE Side Effects

None.

Conventions

- Access (see Section 4.3.2 on page 30).

Chapter 14

Extended 1.0 SMB Miscellaneous Requests

This section defines the remaining elements of the extended 1.0 SMB protocol. They are:

SMBcopy	copy one or more files
SMBecho	test an LMX session
SMBioctl	I/O device control
SMBmove	move one or more files by renaming

14.1 SMBcopy Specification

SMBcopy Detailed Description

This extended 1.0 protocol request copies one or more files from a given path to a new path on a single LMX server. The source path may include wildcards. The destination may be a directory or a single file, but it may not include wildcards. If the destination is a directory, the source file(s) are copied into that directory; if the destination is a regular file, the source file(s) are appended to it (possibly after the destination is truncated).

SMBcopy Deviations

None.

SMBcopy Field Descriptions

From SMB redirector		To SMB redirector	
Field Name	Field Value	Field Name	Field Value
smb_com	*SMBcopy*	*smb_com*	*SMBcopy*
smb_wct	3	*smb_wct*	1
smb_vwv[0]	*smb_tid2*	*smb_vwv*[0]	*smb_cct*
smb_vwv[1]	*smb_ofun*	*smb_bcc*	min=0
smb_vwv[2]	*smb_flags*	*smb_buf*[]	*smb_errfile*
smb_bcc	min=2		
smb_buf[]	*smb_path*		
	smb_new_path		

smb_tid2 The TID corresponding to *smb_new_path*. The TID for *smb_path* is sent in *smb_tid* in the SMB header. If *smb_tid2* is –1, the TID in *smb_tid* should be used for *smb_new_path* as well; this permits *SMBcopy* to be chained to *SMBtconX*.

smb_ofun This is an open function field (see Section 5.3.8 on page 46). If *smb_new_path* is a simple file *smb_ofun* applies at the start of the operation; in the case of wildcards all subsequent files will then be appended. It is applied to each copied file when *smb_new_path* is a directory.

smb_flags This 16 bit field contains a set of flags controlling the copy operations:

Bit 0 If set, the destination must be a file.

Bit 1 If set, the destination must be a directory.

Bit 2 Copy destination mode: 0=binary (indicating the contents of the file are not to be interpreted), 1=ASCII (indicating DOS format text file). This bit is ignored.

Bit 3 Copy source mode: 0=binary (indicating the contents of the file are not to be interpreted), 1=ASCII (indicating DOS format text file). This bit is ignored.

Bit 4 If set, all writes must be verified by comparing the copied destination to the original source(s).

Bit 5 If set, indicates a tree copy is requested. A tree copy means the contents of the directory and any subdirectories are to be copied. This bit only has meaning if the extended 2.0 SMB dialect was negotiated.

All other bits are reserved and should be ignored.

smb_path An ASCIIZ buffer containing the name of the file(s) to be copied; wildcard characters are permitted. The path is interpreted relative to *smb_tid* in the SMB header.

smb_new_path An ASCIIZ buffer containing the name of the destination to which the source file(s) are to be copied. Wildcards may not be used. The path is interpreted relative to *smb_tid2* in the *SMBcopy* subheader.

smb_cct A 16-bit integer containing the actual number of files copied.

smb_errfile This is an ASCIIZ buffer which may contain the name of the source file on which an error was encountered during a copy operation. When a copy error is encountered, the expanded source filename is returned in *smb_errfile* and the error code is returned in *smb_err* (in the SMB header).

SMBcopy Error Code Descriptions

CAE Code	DOS Class	DOS Code	Description
EACCES	ERRDOS	ERRnoaccess	Component of path-prefix denies search permission.
EAGAIN	ERRDOS	ERRshare	There are outstanding record locks on the file.
EEXIST	ERRSRV	ERRfilexists	Destination file exists.
EINTR	ERRSRV	ERRerror	A signal was caught during the open operation.
EISDIR	ERRDOS	ERRnoaccess	Can't copy onto a directory.
EMFILE	ERRSRV	ERRerror	Maximum number of file descriptors are currently open in this process.
ENFILE	ERRDOS	ERRnofids	System file table is full.
ENOENT	ERRDOS	ERRbadfile	File does not exist, or component of pathname does not exist.
ENOSPC	ERRSRV	ERRerror	The system is out of resources necessary to create files.
ENOTDIR	ERRDOS	ERRbadpath	Component of either path-prefix is not a directory.
ENXIO	ERRSRV	ERRerror	One of the TIDs is not for a file system subtree.
EROFS	ERRSRV	ERRerror	Destination file system subtree is read-only.
ETXTBSY	ERRSRV	ERRerror	Can't copy onto programme being executed.
-	ERRSRV	ERRinvnid	Invalid TID.
-	ERRSRV	ERRinvdevice	One of the TIDs is not for a file system subtree.
-	ERRDOS	ERRnofiles	No more files matching the specified criteria.
-	ERRDOS	ERRbadshare	Share conflict when creating a destination file.
-	ERRSRV	ERRbaduid	The UID given (*smb_uid*) is not known as a valid ID on this LMX session.
-	SUCCESS	SUCCESS	Everything worked, no problems.

SMBcopy Preconditions

1. The SMB redirector has sent a valid SMB with a valid *smb_tid* and *smb_tid2* for file system subtrees; the *smb_tid2* resource must allow writes.
2. The SMB redirector has appropriate read/search permission on source and destination paths, and write permission on the destination file or into the destination directory.

SMBcopy Postconditions

Not all files may have been copied; *smb_errfile* will indicate which copy failed.

SMBcopy Side Effects

Some files may be overwritten if *smb_ofun* flags requested it.

Conventions

- Access (see Section 4.3.2 on page 30).
- Filename (see Section 3.5 on page 15).
- Wildcards (see Section 3.6 on page 17).

14.2 SMBecho Specification

SMBecho Detailed Description

This extended protocol request is used to test an LMX session by exchanging messages between the SMB redirector and LMX server. Since it is used to verify communications, the request may be issued at any time during the life of an LMX session, except before an *SMBnegprot* request has been issued, and not while a raw exchange is in progress (for example, *SMBwritebraw*).

The LMX server will respond with the exact number of messages specified in the request.

SMBecho Deviations

None.

SMBecho Field Descriptions

From SMB redirector		To SMB redirector	
Field Name	Field Value	Field Name	Field Value
smb_com	*SMBecho*	*smb_com*	*SMBecho*
smb_wct	1	*smb_wct*	1
smb_vwv[0]	*smb_reverb*	*smb_vwv*[0]	*smb_sequence*
smb_bcc	min=0	*smb_bcc*	min=0
smb_buf[]	*smb_data*	*smb_buf*[]	*smb_data*

smb_reverb A 16-bit integer indicating the number of responses the LMX server should generate for this request. If zero, no response at all will be generated.

smb_data This string of bytes is test data which is specified by the SMB redirector in its request and returned by the LMX server in every response. The string of bytes is not formatted; the LMX server must be careful to exactly reproduce it and set *smb_bcc* correctly in the responses.

smb_sequence A 16-bit integer containing the sequence number of this particular response. The first response would have *smb_sequence* = 1, and the last response would set *smb_sequence* to *smb_reverb*.

SMBecho Error Code Descriptions

CAE Code	DOS Class	DOS Code	Description
-	ERRSRV	ERRnoaccess	LMX session has not been established.
-	ERRSRV	ERRbaduid	The UID given (*smb_uid*) is not known as a valid ID on this LMX session.
-	ERRSRV	ERRnosupport	Requested function is not supported.
-	SUCCESS	SUCCESS	Everything worked, no problems.

No CAE errors are possible.

SMBecho Preconditions

None.

SMBecho Postconditions

None.

SMBecho Side Effects

None.

Conventions

None.

14.3 SMBioctl Specification

SMBioctl Detailed Description

This extended protocol request permits detailed control of I/O devices by the SMB redirector. The actual forms of control available are device-specific and implementation-dependent.

SMBioctl Deviations

Because the mapping between ioctl request numbers and actual functionality varies from implementation to implementation, it is impossible to provide this functionality in a portable manner. Nonetheless, SMB redirectors using the LMX server may generate *SMBioctl* requests.

An LMX server which does not support the *SMBioctl* request should return error code ERRnosupport in error class ERRSRV if it receives such a request.

14.4 SMBmove Specification

SMBmove Detailed Description

This extended protocol request is used to move files between directories on the LMX server. Directories as well as regular files may be moved into a new directory. The *SMBmove* protocol removes the deviations of *SMBmv* and allows for relocating files to different file system subtrees. A move of a directory cannot have a destination located in the directory itself or any subdirectory within the source directory. In these conditions the error <ERRDOS, ERRbadpath> is to be returned.

The source path may include wildcards in the last component of the path, but the destination path must specify a single file or directory (that is, no wildcards). If the destination is a directory, the source file(s) are moved into that directory; if the destination is a regular file, all source files but the last one are lost, and the last one is renamed to the destination path. The sequence in which files match a wildcard specification is undefined, so the specific file which will be given the destination name cannot be specified.

SMBmove Deviations

None.

SMBmove Field Descriptions

From SMB redirector		To SMB redirector	
Field Name	Field Value	Field Name	Field Value
smb_com	*SMBmove*	*smb_com*	*SMBmove*
smb_wct	3	*smb_wct*	1
smb_vwv[0]	*smb_tid2*	*smb_vwv*[0]	*smb_count*
smb_vwv[1]	*smb_ofun*	*smb_bcc*	min=0
smb_vwv[2]	*smb_flags*	*smb_buf*[]	*smb_errfile*
smb_bcc	min=2		
smb_buf[]	*smb_path*		
	smb_new_path		

smb_tid2 The TID corresponding to *smb_new_path*. The TID for *smb_path* is sent in *smb_tid* in the SMB header. If *smb_tid2* is –1, the TID in *smb_tid* should be used for *smb_new_path* as well; this permits *SMBmove* to be chained to *SMBtconX*.

smb_ofun This is an open function field (see Section 5.3.8 on page 46). If *smb_new_path* is a simple file *smb_ofun* applies at the start of the operation; in the case of wildcards all subsequent files will then be appended. It is applied to each moved file when *smb_new_path* is a directory.

smb_flags This 16 bit field contains a set of flags controlling the copy operations:

Bit 0 If set, the destination must be a file.

Bit 1 If set, the destination must be a directory.

Bit 4 If set, all writes must be verified by comparing the copied destination to the original source(s).

All other bits are reserved and should be ignored.

smb_path An ASCIIZ buffer containing the name of the file(s) to be moved; wildcard characters are permitted. The path is interpreted relative to *smb_tid* in the SMB header.

smb_new_path An ASCIIZ buffer containing the name of the destination to which the source file(s) are to be copied. Wildcards may not be used. The path is interpreted relative to *smb_tid2* in the *SMBmove* subheader.

smb_count A 16-bit integer containing the actual number of files moved.

smb_errfile This is an ASCIIZ buffer which may contain the name of the source file on which an error was encountered, the expanded source filename is returned in *smb_errfile* and the error code is returned in *smb_err* (in the SMB header).

SMBmove Error Code Descriptions

CAE Code	DOS Class	DOS Code	Description
EACCES	ERRDOS	ERRnoaccess	Search permission is denied on a component of either path-prefix.
EACCES	ERRDOS	ERRnoaccess	No write access to destination directory.
EEXIST	ERRDOS	ERRfilexists	Directory or file already exists.
EINTR	ERRSRV	ERRerror	A signal was caught during a system call.
EMLINK	ERRSRV	ERRerror	Maximum number of links to a file would be exceeded.
ENOENT	ERRDOS	ERRbadfile	A component of either path-prefix does not exist, *smb_path* does not exist, or *smb_new_path* is a null string.
ENOSPC	ERRSRV	ERRerror	Directory containing the link cannot be extended.
ENOTDIR	ERRDOS	ERRbadpath	A component of either path-prefix is not a directory.
EROFS	ERRSRV	ERRnoaccess	Read-only file system.
EXDEV	ERRDOS	ERRnoaccess	*smb_path* and *smb_new_path* are on different logical devices.
-	ERRDOS	ERRnofiles	No files match *smb_path*.
-	ERRDOS	ERRbadshare	Share conflict when creating or appending to a destination file.
-	ERRSRV	ERRerror	Corrupt SMB request.
-	ERRSRV	ERRinvtid	Invalid TID.
-	ERRSRV	ERRnosupport	Requested function is not supported.
-	ERRSRV	ERRaccess	The resource represented by the TID does not allow writes.
-	ERRSRV	ERRbaduid	The UID given (*smb_uid*) is not known as a valid ID on this LMX session.
-	SUCCESS	SUCCESS	No errors.

SMBmove Preconditions

1. The SMB redirector has sent a valid SMB request; both TIDs are for file system subtrees; the SMB redirector has delete permission under the source TID and create permission under the destination TID.
2. The source file(s) or directory must exist.
3. Files must not be open by other SMB redirectors. If they are, the error <ERRDOS, ERRbadshare> is returned.
4. The SMB redirector has write permission in the destination directory and delete (write) permission in the source directory.

SMBmove Postconditions

1. If the move succeeded, none of the matching source files can be found under the old names, and the files are now accessible under the new names.
2. If a move fails, the reason for the failure is returned in *smb_errfile*, along with an error return. No remaining moves are attempted, and *smb_count* reflects the actual number of files moved.

SMBmove Side Effects

Moves of multiple files to a single regular file result in the loss of all but the last file.

Conventions

- Access (see Section 4.3.2 on page 30).
- Filenames (see Section 3.5 on page 15).
- Wildcards (see Section 3.6 on page 17).

Chapter 15

Extended 2.0 Protocol Additions and Modifications

This chapter documents the changes and additions to the extended 1.0 dialect that take effect when the extended 2.0 dialect is negotiated. These SMBs and the *SMBtrans2* (refer to Chapter 16 on page 207) constitute the additions to the extended 1.0 dialect for the extended 2.0 dialect. There is no affect on the *SMBnegprot* protocol for the extended 2.0 protocol. Refer to the extended 1.0 protocol description for details.

15.1 SMBsesssetupX Specification

SMBsesssetupX Detailed Description

This extended 2.0 protocol request is used to further set up the LMX session normally just established via the *SMBnegprot* request/response. The *SMBsesssetupX* request serves one additional purpose over the activities performed in the extended 1.0 dialect. That purpose is to allow the SMB redirector system to challenge the LMX server with an encryption key. The LMX server must use the encryption key to return a response. Based on the response value, the SMB redirector can determine whether the LMX server is really the LMX server desired or an imposter.

- User Identification

  The actual semantics for this request are governed by the security mode of the LMX server. See Section 3.3 on page 12 for a discussion of these modes.

  In user-level security mode, the SMB redirector will establish a mapping between a particular username on the LMX server and a UID which the SMB redirector will use to represent that user. A password may be sent by the SMB redirector to authenticate that the person using the SMB redirector is indeed the username to be mapped to. Further, the password may be encrypted to ensure security.

  The LMX server validates the name and password supplied and, if valid, it generates a UID corresponding to the specified username. That actual UID will be sent in all subsequent requests by the SMB redirector and used by the LMX server for access checks required by requests.

  The value of the UID is relative to an LMX session; it is possible for the same UID value to represent two different users on two different LMX sessions on the LMX server. The LMX server must map the pair of <LMX session ID, UID> to the different accounts. In share-level security mode, the username and password are not used. The LMX server should use a unique, reserved account and corresponding UID to perform access checks for all requests.

- SMB redirector Communications Parameters

  The LMX server, in its response to the *SMBnegprot* request, has set some parameters for the communication it was expecting from the SMB redirector. In the *SMBsesssetupX* request, the SMB redirector indicates the parameters for the communication it is expecting from the LMX server. These values may be different; for example, the LMX server may be able to receive a maximum message size of 16K bytes, while the SMB redirector can only receive 1K bytes.

  Some LMX servers may need to renegotiate buffer sizes after the *SMBsesssetupX* exchange. This renegotiation is available through the *SMBtcon* request, but not through *SMBtconX*.

SMBsesssetupX Deviations

None.

SMBsesssetupX Field Descriptions

From SMB redirector		To SMB redirector	
Field Name	Field Value	Field Name	Field Value
smb_com	*SMBsesssetupX*	*smb_com*	*SMBsesssetupX*
smb_wct	10	*smb_wct*	3
smb_vwv[0]	*smb_com2*	*smb_vwv*[0]	*smb_com2*
smb_vwv[1]	*smb_off2*	*smb_vwv*[1]	*smb_off2*
smb_vwv[2]	*smb_bufsize*	*smb_vwv*[2]	*smb_action*
smb_vwv[3]	*smb_mpxmax*	*smb_bcc*	Minimum = 0
smb_vwv[4]	*smb_vc_num*	*smb_buf*[]	*smb_encresp*[]
smb_vwv[5,6]	*smb_sesskey*		
smb_vwv[7]	*smb_apasslen*		
smb_vwv[8]	*smb_encryptlen*		
smb_vwv[9]	*smb_encryptoff*		
smb_bcc	min val=0		
smb_buf[]	*smb_apasswd*		
	smb_aname		

smb_com2 Description can be found in Section 3.9 on page 22.

smb_off2 Description can be found in Section 3.9 on page 22.

smb_bufsize The size of the largest message the SMB redirector is willing to receive. It must be true that *smb_bufsize* ≤ *smb_maxxmt* (see Section 6.1 on page 55).

smb_mpxmax The maximum number of requests which the SMB redirector will have outstanding on a single LMX session. It must be true that *smb_mpxmax* ≤ *smb_maxmux* (see Section 6.1 on page 55).

smb_vc_num Permits multiple NetBIOS sessions to be associated with a single LMX session. If zero (0), this NetBIOS session is the first or only NetBIOS session associated with the NetBIOS session being set up. If *smb_vc_num* is zero (0) and there are other previously established NetBIOS session still connected from this SMB redirector, it is recommended that the LMX server abort the previous NetBIOS session and free up the resources held.

smb_sesskey A 32-bit integer which identifies to which LMX session this NetBIOS session is associated. Ignored when *smb_vc_num* is zero (0). This value would be obtained from the *smb_sesskey* field in the response to the *SMBnegprot* associated with the LMX session this NetBIOS session is to be made a part of.

smb_apasslen Length of the *smb_apasswd* field.

smb_encryptlen The size of the encryption key used to challenge the LMX server.

smb_encryptoff The byte offset from the start of the SMB header to the encryption key.

smb_encresp[] The LMX server response to the encryption key challenge from the SMB redirector.

smb_apasswd A character string containing the password, possibly encrypted. Ignored by an LMX server in share-level security mode.

smb_aname An ASCIIZ (not type 04) buffer containing the username to be associated with *smb_uid* and validated with *smb_apasswd*. Ignored by an LMX server in share-level security mode. The length of this field is derived from the difference between *smb_bcc* and *smb_apasslen*.

smb_action A bit-encoded field indicating the results of a successful LMX session setup. If bit 0 is clear, everything went normally. If bit 0 is set, the LMX session was setup but a default or guest account was used instead of an individual account represented by the username provided. (An LMX server in share-level security mode would set this bit.)

SMBsesssetupX Error Code Descriptions

CAE Code	DOS Class	DOS Code	Description
-	ERRSRV	ERRerror	Internal LMX server error.
-	ERRSRV	ERRbadpw	Username/password pair was invalid.
-	ERRSRV	ERRtoomanyuids	The LMX server does not support this many UIDs in one LMX session.
-	ERRSRV	ERRerror	No *SMBnegprot* request has been issued on this NetBIOS session.
-	ERRSRV	ERRnosupport	This request cannot be chained to the request which precedes it in this message.
-	SUCCESS	SUCCESS	Everything worked, no problems.

SMBsesssetupX Preconditions

1. The process attempting to secure an LMX session must have established an LMX session with the LMX server and negotiated an extended dialect.
2. The username and password must both be valid instances of those types.
3. *smb_com2* must be a legal chained command.
4. There are many other preconditions based upon the SMBs that may be chained. These are enumerated in the specifications for those SMBs.

SMBsesssetupX Postconditions

1. If there are no errors the UID is valid to be used in future SMBs.
2. There are many other postconditions based upon the SMBs that may be chained. These are enumerated in the specifications for these SMBs.

SMBsesssetupX Side Effects

Conversion of paths to a canonical pathname is controlled by bit 4 of the *smb_flg* flag in the header of this request (see Section 5.1 on page 37).

Conventions

- Opportunistic Locking (see Section 3.8.2 on page 20).
- Chaining (see Section 3.9 on page 22).

The SMBs which may be chained after *SMBsesssetupX* are:

SMBchkpath	*SMBfunique*	*SMBopen*	*SMBsearch*	*SMBtconX*
SMBcopy	*SMBgetatr*	*SMBopenX*	*SMBsetatr*	*SMBunlink*
SMBcreate	*SMBmkdir*	*SMBrename*	*SMBsplopen*	*SMBtrans*
SMBdskattr	*SMBmknew*	*SMBrmdir*	*SMBsplretq*	NIL
SMBffirst	*SMBmv*			

15.2 SMBcopy Specification

SMBcopy Detailed Description

The *SMBcopy* protocol for the extended 2.0 dialect is unchanged from the extended 1.0 dialect except that the request may now be used to specify a copy of entire directory subtrees (tree copy) on the LMX server. The tree copy mode is selected by setting bit 5 of the *smb_flags* field in the *SMBcopy* request (reference bit 5 in **SMBcopy Field Descriptions** on page 187). When the tree copy option is selected the destination must not be an existing file and the source mode must be binary. A request with bit 5 of the *smb_flags* field set and either bit 0 or bit 3 set is not allowed and the LMX server returns the error code <ERRDOS, ERRbadfile>. When the tree copy mode is selected the *smb_cct* field of the response protocol is undefined.

15.3 SMBfindnclose Specification

SMBfindnclose Detailed Description

The *SMBfindnclose* protocol closes the association between a directory handle returned following a resource monitor established using an *SMBtrans2(FINDNOTIFYFIRST)* request to the LMX server and the resulting system directory monitor. This request allows the LMX server to free any resources held in support of the open handle.

SMBfindnclose Field Descriptions

From SMB redirector		To SMB redirector	
Field Name	Field Value	Field Name	Field Value
smb_com	*SMBfindnclose*	*smb_com*	*SMBfindnclose*
smb_wct	1	*smb_wct*	0
smb_vwv[0]	*smb_handle*	*smb_bcc*	0
smb_bcc	0	*smb_bcc*	0

smb_handle The directory handle associated with a previous *SMBtrans2(TRANSACT2_FINDNOTIFYFIRST)*.

SMBfindnclose Error Code Descriptions

CAE Code	DOS Class	DOS Code	Description
-	ERRDOS	ERRbadfid	The SMB redirector has supplied an invalid directory handle.
-	ERRSRV	ERRinvnid	TID specified in command is invalid.
-	ERRSRV	ERRerror	Other CAE error.
-	SUCCESS	SUCCESS	Operation succeeded.

SMBfindnclose Preconditions

None.

SMBfindnclose Postconditions

If the directory handle was valid, it is made invalid and resources used to support the directory search operations have been freed.

SMBfindnclose Side Effects

None.

Conventions

None.

15.4 SMBfindclose Specification

SMBfindclose Detailed Description

The *SMBfindclose* protocol closes the association between a search handle returned following a successful *SMBtrans2(TRANSACT2_FINDFIRST)* request to the LMX server and the resulting system file search. This request allows the LMX server to free any resources held in support of the open handle.

SMBfindclose Field Descriptions

From SMB redirector		To SMB redirector	
Field Name	Field Value	Field Name	Field Value
smb_com	*SMBfindclose*	*smb_com*	*SMBfindclose*
smb_wct	1	*smb_wct*	0
smb_vwv[0]	*smb_handle*	*smb_bcc*	0
smb_bcc	0	*smb_bcc*	0

smb_handle The directory handle associated with a previous *SMBtrans2(TRANSACT2_FINDNOTIFYFIRST)*.

SMBfindclose Error Code Descriptions

CAE Code	DOS Class	DOS Code	Description
-	ERRDOS	ERRbadfid	The SMB redirector has supplied an invalid directory handle.
-	ERRSRV	ERRinvnid	TID specified in command is invalid.
-	ERRSRV	ERRerror	Other CAE error.
-	SUCCESS	SUCCESS	Operation succeeded.

SMBfindclose Preconditions

None.

SMBfindclose Postconditions

If the directory handle was valid, it is made invalid and resources used to support the directory search operations have been freed.

SMBfindclose Side Effects

None.

Conventions

None.

15.5 SMBuloggoffX Specification

SMBuloggoffX Detailed Description

This protocol is used to logoff the user (identified by the UID value in *smb_uid*) previously logged on via the *SMBsesssetupX* protocol.

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

Another *SMBsesssetupX* must be sent in order to reenstate the user on the LMX session.

LMX session termination also causes the UIDs registered on the LMX session to be invalidated. When the LMX session is reestablished, *SMBsesssetupX* request must again be used to validate each user.

The only valid protocol that can be chained in an *SMBuloggoffX* is *SMBsessetupX*.

SMBuloggoffX Field Descriptions

From SMB redirector		To SMB redirector	
Field Name	Field Value	Field Name	Field Value
smb_com	*SMBuloggoffX*	*smb_com*	*SMBuloggoffX*
smb_wct	2	*smb_wct*	2
smb_vwv[0]	*smb_com2*	*smb_vwv*[0]	*smb_com2*
smb_vwv[1]	*smb_off2*	*smb_vwv*[1]	*smb_off2*
smb_bcc	0	*smb_bcc*	0

smb_com2 The secondary command value.

smb_off2 Offset from start of the SMB header to the secondary command.

SMBuloggoffX Error Code Descriptions

CAE Code	DOS Class	DOS Code	Description
-	ERRSRV	ERRinvnid	TID specified in command is invalid.
-	ERRSRV	ERRerror	Other CAE error.
-	ERRSRV	ERRbaduid	The UID given (*smb_uid*) is not known as a valid ID on this LMX session.
-	SUCCESS	SUCCESS	Operation succeeded.

SMBuloggoffX Preconditions

None.

SMBuloggoffX Postconditions

If the user was previously logged on, his logon identity as specified in the *SMBsesssetupX* is removed, but the LMX session remains.

SMBulogoffX Side Effects

Another *SMBsesssetupX* must be sent to log the user into the LMX server.

Conventions

None.

Chapter 16

Extended 2.0 Protocol SMBtrans2

The *SMBtrans2* protocol is used to extend the original file-sharing protocols with extended attribute and long filename support. An FID obtained from the new requests may be used in previously defined SMB requests and *vice versa*.

The format of enhanced and new commands is defined commencing at the *smb_wct* field. All messages will include the standard SMB header defined in Section 5.1 on page 37. When an error is encountered, an LMX server may choose to return only the header portion of the response (i.e., *smb_wct* and *smb_bcc* both contain zero).

16.1 SMBtrans2

16.1.1 Request Formats

Transaction SMB Request Formats			
Primary Request		Secondary Request	
Field Name	Field Value	Field Name	Field Value
smb_com	*SMBtrans2*	*smb_com*	*SMBtrans2*
smb_wct	14+*smb_suwcnt*	*smb_wct*	8
smb_vwv[0]	*smb_tpscnt*	*smb_vwv*[0]	*smb_tpscnt*
smb_vwv[1]	*smb_tdscnt*	*smb_vwv*[1]	*smb_tdscnt*
smb_vwv[2]	*smb_mprcnt*	*smb_vwv*[2]	*smb_pscnt*
smb_vwv[3]	*smb_mdrcnt*	*smb_vwv*[3]	*smb_psoff*
smb_vwv[4]	*smb_msrcnt*	*smb_vwv*[4]	*smb_psdisp*
smb_vwv[5]	*smb_flags*	*smb_vwv*[5]	*smb_dscnt*
smb_vwv[6-7]	*smb_timeout*	*smb_vwv*[6]	*smb_dsoff*
smb_vwv[8]	*smb_rsvd1*	*smb_vwv*[7]	*smb_dsdisp*
smb_vwv[9]	*smb_pscnt*	*smb_vwv*[8]	*smb_fid*
smb_vwv[10]	*smb_psoff*	*smb_bcc*	
smb_vwv[11]	*smb_dscnt*		*smb_param*
smb_vwv[12]	*smb_dsoff*		*smb_data*
smb_vwv[13]	*smb_suwcnt*		
smb_vwv[14-]	*smb_setup*[]		
smb_bcc			
smb_buf[]	*smb_name*		
	smb_param		
	smb_data		

smb_tpscnt A 16-bit unsigned integer containing the total number of parameter bytes being sent. This value may be revised downward in any or all secondary requests. The smallest value of *smb_tpscnt* sent during this transaction must equal the sum of all the *smb_pscnt* fields in all requests sent during the transaction.

smb_tdscnt A 16-bit unsigned integer containing the total number of data bytes being sent. This value may be revised downward in any or all secondary requests. The smallest value of *smb_tdscnt* sent during this transaction must equal the sum of all the *smb_dscnt* fields in all requests sent during the transaction.

smb_mprcnt A 16-bit integer containing the maximum number of parameter bytes the SMB redirector expects to be returned. The LMX server may not exceed this limit in its response.

smb_mdrcnt A 16-bit unsigned integer containing the maximum number of data bytes the SMB redirector expects to be returned. The LMX server may not exceed this limit in its response.

smb_msrcnt A 16-bit integer containing the maximum number of setup fields the SMB redirector expects to be returned. The LMX server may not exceed this limit in its response. The value of *smb_msrcnt* must be less than or equal to 255 and is stored in the low-order byte of the field; the high-order byte is reserved and must be zero.

smb_flags A 16-bit field containing flags altering the behaviour of the request. The flags are:

Bit 0 If set, the TID on which this transaction was requested is closed after the transaction is completed.

Bit 1 If set, the transaction is one way; that is, no final response should be generated by the LMX server. An interim response, if required by the flow of the transaction, should be produced regardless of the setting of this bit.

Bits 2-15 Reserved; MBZ.

smb_timeout A 32-bit integer specifying the number of milliseconds to wait for completion of the requested operation before causing a timeout. A value of zero (0) means no delay (that is, do not queue the request). A value of –1 indicates to wait forever. See Section 3.11 on page 25.

smb_rsvd1 A 16-bit reserved field which must be zero.

smb_pscnt A 16-bit unsigned integer indicating the number of parameter bytes being sent in this particular request; i.e., the size of *smb_param*.

smb_psoff A 16-bit integer giving the offset, in bytes, from the start of the SMB header to the beginning of the *smb_param* field. This permits *smb_param* to be preceded in the request by pad bytes to result in better alignment of the buffer.

smb_psdisp A 16-bit integer giving the absolute displacement amongst all parameter bytes for this transaction for the parameter bytes contained in this request. This is used by the LMX server to correctly assemble all the parameter bytes received even if the requests were received out of sequence.

smb_dscnt A 16-bit unsigned integer indicating the number of data bytes being sent in this particular request; i.e., the size of *smb_data*.

smb_dsoff A 16-bit integer giving the offset, in bytes, from the start of the SMB header to the beginning of the *smb_data* field. This permits *smb_data* to be preceded in the request by pad bytes to result in better alignment of the buffer.

smb_dsdisp A 16-bit integer giving the displacement amongst all data bytes for this transaction of the data bytes contained in this request. This is used by the LMX server to correctly assemble all the data bytes received even if the requests were received out of sequence.

smb_fid A 16-bit integer containing the FID for file-based requests. Otherwise the value is 0xffff.

smb_suwcnt A 16-bit integer containing the number of setup 16-bit fields sent in the primary request. This value must be less than or equal to 255 and is stored in the low-order byte of the 16-bit field; the high-order value is reserved and must be zero.

smb_setup[] An array of 16-bit fields of setup data.

smb_bcc Contains the total size in bytes of the data to follow, including any pad bytes added for alignment. The length of this array is given by *smb_swcnt* and may be zero.

smb_name A null-terminated ASCIIZ string containing the transaction name. No pad bytes are permitted before this field; it must immediately follow the *smb_bcc* field.

smb_param An array of bytes, beginning at *smb_psoff* bytes into the request and containing *smb_pscnt* bytes. Padding may precede this field, as *smb_psdisp* points to its beginning; for the same reason, *smb_param* is not required to precede *smb_data* in each message.

smb_data An array of bytes, beginning at *smb_dsoff* bytes into the request and containing *smb_dscnt* bytes. Padding may precede this field, as *smb_dsdisp* points to its beginning; for the same reason, this field is not always required to follow *smb_param*.

16.1.2 Response Format

Transaction SMB Response Formats			
Interim Response		Final Response	
Field Name	Field Value	Field Name	Field Value
smb_com	*SMBtrans2*	*smb_com*	*SMBtrans2*
smb_wct	0	*smb_wct*	10+*smb_suwcnt*
smb_bcc	0	*smb_vwv*[0]	*smb_tprcnt*
		smb_vwv[1]	*smb_tdrcnt*
		smb_vwv[2]	*smb_rsvd*
		smb_vwv[3]	*smb_prcnt*
		smb_vwv[4]	*smb_proff*
		smb_vwv[5]	*smb_prdisp*
		smb_vwv[6]	*smb_drcnt*
		smb_vwv[7]	*smb_droff*
		smb_vwv[8]	*smb_drdisp*
		smb_vwv[9]	*smb_suwcnt*
		smb_vwv[10-]	*smb_setup*
		smb_bcc	
			smb_param
			smb_data

The meaning of the parameters is identical to the definitions above with the parameter names changed; for example, *smb_tprcnt* is the total number of parameter bytes being returned, and is used in the same way as *smb_tpscnt* in the request messages.

As was the case in the request messages, the ordering of *smb_param* and *smb_data* is not required, since *smb_prdisp* and *smb_drdisp* are sufficient to locate each correctly.

16.1.3 Transaction Flow

A small set of rules governs the flow of the various protocol elements making up a transaction, including which request or response type to send at any particular time.

1. The SMB redirector sends the first (primary) request which identifies the total bytes (parameters and data) which are to be sent, and contains the setup 16-bit fields, and as many of the parameter and data bytes as will fit in the maximum negotiated buffer size. This request also identifies the maximum number of bytes (setup, parameters and data) the LMX server may return when the transaction is completed. The parameter bytes are immediately followed by the data bytes (the length fields identify the break point). If all the bytes fit in the single buffer, skip to step 4.
2. The LMX server responds with a single interim response meaning O.K., send the remainder of the bytes, or (if error response) terminate the transaction.
3. The SMB redirector then sends a secondary request full of bytes to the LMX server. This step is repeated until all bytes have been delivered to the LMX server.
4. The LMX server sets up and performs the transaction with the information provided.
5. Upon completion of the transaction, if bit 1 of *smb_flag* was not set in the primary request, the LMX server sends back up to the number of parameter and data bytes requested (or as many as will fit in the negotiated buffer size). This step is repeated until all bytes requested have been returned. Fewer than the requested number of bytes (from *smb_mdrcnt* and *smb_mprcnt*) may be returned.

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

SMB redirector	→	*SMBtrans2* request (data)	>>	LMX server
SMB redirector	<<-	OK send remaining data	<-	LMX server
SMB redirector	→	*SMBtrans2* secondary request 1 (data)	>>	LMX server
SMB redirector	→	*SMBtrans2* secondary request 2 (data)	>>	LMX server
SMB redirector	→	*SMBtrans2* secondary request *n* (data) (LMX server sets up and performs the *SMBtrans2*)	>>	LMX server
SMB redirector	<<-	*SMBtrans2* response 1 (data)	<-	LMX server
SMB redirector	<<-	*SMBtrans2* response 2 (data)	<-	LMX server
SMB redirector	<<-	*SMBtrans2* response *n* (data)	<-	LMX server

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

SMB redirector	→	*SMBtrans2* request (data) (LMX server sets up and performs the *SMBtrans2*)	>>	LMX server
SMB redirector	<<-	*SMBtrans2* response 1 (data) (only one if all data fit in buffer)	<-	LMX server
SMB redirector	<<-	*SMBtrans2* response 2 (data)	<-	LMX server
SMB redirector	<<-	*SMBtrans2* response *n* (data)	<-	LMX server

Note that the primary request through to the final response make up the complete protocol: thus, the TID, PID, UID and MID are expected to remain constant and can be used by both the LMX server and SMB redirector to route the individual messages of the protocol to the correct process. Also, it is the responsibility of the LMX server to assemble the multiple requests into the final complete request to execute. Similarly, the SMB redirector will assemble the response sequence.

The simplest form of an *SMBtrans2* is to send a single primary request and (optionally) receive a single, final response.

16.1.4 Service

The *SMBtrans2* protocol allows transfer of parameter and data blocks greater than the maximum negotiated buffer size between the SMB redirector and the LMX server.

The *SMBtrans2* command scope includes (but is not limited to) IOCTL device requests and file system requests which require the transfer of an extended attribute list.

The *SMBtrans2* protocol is used to transfer a request for any of a set of supported functions on the LMX server which may require the transfer of large data blocks. The function requested is identified by the first 16-bit field in the *SMBtrans2 smb_setup*[] field. Other function-specific information may follow the function identifier in the *smb_setup*[] or in the *smb_param* fields. The functions supported are not defined by the protocol, but by SMB redirector and LMX server implementations. The protocol simply provides a means of delivering them and retrieving the results.

The number of bytes needed in order to perform the *SMBtrans2* request may be more than will fit in the negotiated buffer size.

At the time of the request, the SMB redirector knows the number of parameter and data bytes expected to be sent and passes this information to the LMX server in the primary request fields *smb_tpscnt* and *smb_tdscnt*. This may be reduced by lowering the total number of bytes expected (*smb_tpscnt* and/or *smb_tdscnt*) in the secondary request.

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

Likewise, when the amount of data bytes received (total of each *smb_dscnt*) equals the total amount of data bytes expected (smallest *smb_tdscnt*), then the LMX server has received all the data bytes.

The parameter bytes should normally be sent first, followed by the data bytes. However, the LMX server knows where each begins and ends in each buffer by the offset fields (*smb_psoff* and *smb_dsoff*) and the length fields (*smb_pscnt* and *smb_dscnt*). The displacement of the bytes is also known (*smb_psdisp* and *smb_dsdisp*). Thus the LMX server is able to reassemble the parameter and data bytes regardless of the order sent by the SMB redirector.

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

The SMB redirector knows the maximum amount of data and parameter bytes the LMX server may return from *smb_mprcnt* and *smb_mdrcnt* of the request. The LMX server informs the SMB redirector of the actual amounts being returned in each buffer of the response in the fields *smb_tprcnt* and *smb_tdrcnt*.

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

When the amount of parameter bytes received (total of each *smb_prcnt*) equals the total amount of parameter bytes expected (smallest *smb_tprcnt*), then the SMB redirector has received all the parameter bytes.

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

The parameter bytes should normally be returned first, followed by the data bytes. However, the SMB redirector knows where each begins and ends in each buffer by the offset fields (*smb_proff* and *smb_droff*) and the length fields (*smb_prcnt* and *smb_drcnt*). The displacement of the bytes relative to the start of each response is also known (*smb_prdisp* and *smb_drdisp*). Thus the SMB redirector is able to reassemble the parameter and data bytes regardless of the order the information is returned.

16.1.5 Extended Attribute

An overview of EAs was given in Section 4.3.7 on page 31. The extended 2.0 SMB dialect allows for the creation, viewing and manipulation of EAs. Support for EAs is optional and it is possible for an LMX server to negotiate the extended 2.0 protocol dialect and not support EAs. In this case, a null EA list is provided on all *SMBtrans2* requests that return EAs and the error <ERRDOS, ERROR_EAS_NOT_SUPPORTED> is returned.

A null EA list is a zero'ed FEA structure (defined below), or in other words, four zero bytes.

16.1.5.1 Errors Encountered When Creating EAs

An LMX server is not required to support EAs when the extended 2.0 dialect is selected. If the LMX server does not support EAs, the error <ERRDOS, ERROR_EAS_NOT_SUPPORTED> will be returned when the SMB redirector attempts to set EAs on a file and a null EA list will be returned when EAs are requested by the SMB redirector. In the case where EAs are supported, when the LMX server is attempting to store EAs sent during the creation of the file and it is not possible to store the EAs due to memory restrictions or file system space, the error code <ERRSRV, ERRerror> or the error code <ERRSRV, ERRnoresources> may be returned. In this case, the creation of the file will fail and no FID will be returned to the SMB redirector.

16.1.5.2 Encapsulation of EAs in the SMB Protocol

There are two forms of structures that may be returned when passing EAs in the SMB protocol. The first is the full extended attribute structure, or FEA structure, and the second is a shorter form for getting the extended attribute names available, or the GEA structure. The GEA structure is used only in SMB requests. FEA structures are used in both SMB requests and responses.

Extended attributes are carried in the SMB requests and responses in these FEA and GEA structures. To contain multiple EAs a "list" structure is used. Both the FEA and GEA structures are encapsulated in this list structure. The list structure is a 32-bit integer size value followed by the FEA or GEA structure. This size value includes its own field length and is the total length of all contained structures in the list.

16.1.5.3 FEA Structure

The FEA structure contains the values for extended attributes (EAs) on a file. An extended attribute is a "name","value" pair where the name is an ASCIIZ string and the value is an unformatted binary area. It is up to the user application to impose format on the value information. This structure is used to carry EAs inside the SMB protocol. When the text below references an EA list inside the protocol, this is the structure containing the user-defined EA.

The "name","value" pair is represented by the following structure:

Name	Description
fEA	A single byte that specifies EA flags. The only flag defined at this time is FEA_NEEDEA which is equal to 0x80. When set to 1, the FEA_NEEDEA flag indicates that EAs are needed on the file.
cbNameLen	A single byte that specifies the length of the EA name not including the null-terminating character.
cbValueLen	A 16-bit unsigned integer specifying the length of the EA value.
cbName[ ]	Zero-terminated string of cbNameLen+1 bytes. This data immediately follows the *cbValueLen* field.
cbValue[ ]	Variable number of EA value bytes. This data immediately follows the *cbName*[ ] field.

The encapsulated FEA list as it is stored in the SMB protocol is illustrated below.

FEA Length
(32-bit integer)

Flag
8 bit

Name Length
8 bit

Value Length
16 bit

Null-terminated name
.
.

Value data
.
.

Flag
8 bit

Name Length
8 bit

Value Length
16 bit

Null-terminated name
.
.

Value data
.
.

As can be seen above, a null FEA list has a length value of 8 followed by a zero flags field, a zero name length and a zero value length.

16.1.5.4 GEA Structure

The GEA structure contains the names for EAs on a file. An EA name is an ASCIIZ string.

The EA name is represented by the following structure:

Name	Description
cbNameLen	A single byte that specifies the length of the EA name not including the null-terminating character.
cbName[ ]	The byte location of the name. This name immediately follows the *cbNameLen* field.

The encapsulated GEA list is shown below as it is stored in the SMB protocol.

GEA Length (32-bit integer)
Name Length 8 bit
Null-terminated name
.
Name Length 8 bit
Null-terminated name
.

16.1.6 Information Levels

Many of the extended 2.0 protocols have an information level passed as an argument. This information level is described here. The information level controls the amount and type of information on a file that is returned to the SMB redirector. The information level has the following valid values and meanings:

1 DOS-compatible. This returns information in a manner consistant with DOS or the other dialect levels. Specifically, no extended attribute information is returned to the SMB redirector.

2 This value indicates that the size of the complete extended attribute list (that is, name and value pair) is to be returned to the SMB redirector in an EA encapsulating structure, but the FEA list is not included. This is performed by including a null FEA list (that is, all sizes zero) in the *smb_data* field of the response.

3 This indicates that the complete collection of FEA structures contained in an EA encapsulating structure is to be returned to the SMB redirector. The FEA structures returned are stored in the *smb_data* field of the SMB response.

16.1.7 Defined SMBtrans2 Protocols

This section specifies the defines used by the *SMBtrans2* protocol.

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

Manifest	Value	Meaning
TRANSACT2_OPEN	0x00	Open or create a file.
TRANSACT2_FINDFIRST	0x01	Find the first file in a directory.
TRANSACT2_FINDNEXT	0x02	Continue search of a directory.
TRANSACT2_QFSINFO	0x03	Query information about a file system.
TRANSACT2_SETFSINFO	0x04	Set information on a file system.
TRANSACT2_QPATHINFO	0x05	Query information about a special file or directory.
TRANSACT2_SETPATHINFO	0x06	Set information on a special file or directory.
TRANSACT2_QFILEINFO	0x07	Query information about a file.
TRANSACT2_SETFILEINFO	0x08	Set information on a file.
TRANSACT2_FINDNOTIFYFIRST	0x0b	Commence monitoring changes on a file or directory.
TRANSACT2_FINDNOTIFYNEXT	0x0c	Continue monitoring changes on a file or directory.
TRANSACT2_MKDIR	0x0d	Create a directory.

16.2 TRANSACT2_OPEN

The function code *TRANSACT2_OPEN* in *smb_setup*[0] in the primary *SMBtrans2* requests identifies a request to open or create a file with extended attributes.

Primary Request Format

smb_wct	Value = 15.
smb_tpscnt	Total number of parameter bytes being sent.
smb_tdscnt	Total size of extended attribute list.
smb_mprcnt	Maximum return parameter length.
smb_mdrcnt	Value = 0. No data returned.
smb_msrcnt	Value = 0. No setup fields to return.
smb_flags	Bit 0 and bit 1 must be zero.
smb_timeout	Maximum milliseconds to wait for resource to open.
smb_rsvd1	Reserved. Must be zero.
smb_pscnt	Value = tpscnt. Parameters must be in primary request.
smb_psoff	Offset from the start of an SMB header to the parameter bytes.
smb_dscnt	Number of data bytes being sent in this buffer.
smb_dsoff	Offset from the start of an SMB header to the data bytes.
smb_suwcnt	Value = 1.
smb_setup[0]	Value = *TRANSACT2_OPEN*.
smb_bcc	Total bytes following including pad bytes.
smb_param[]	The parameter block for the the *TRANSACT2_OPEN* function is the open-specific information in the following format:

Location	Name	Meaning	
smb_param[0-1]	*open_flags2*	Bit 0	If set, return additional information.
		Bit 1	If set, set single user total file lock (if only access).
		Bit 2	If set, the LMX server should notify the SMB redirector on any action which can modify the file (*SMBunlink*, *SMBsetatr*, *SMBmv*, etc.). If not set, the LMX server need only notify the SMB redirector on another open request.
		Bit 3	If set, return total length of EAs for the file.
smb_param[2-3]	*open_mode*	File open mode. Reference Section 5.3.5 on page 44.	
smb_param[4-5]	*open_sattr*	The set of attributes that the file must have in order to be found while searching to see if it exists. Regardless of the contents of this field, normal files always match.	
smb_param[6-7]	*open_attr*	File attributes (for create). Reference Section 5.3.3 on page 43	
smb_param[8-11]	*open_time*	Create time. Reference Section 5.3.1 on page 43	
smb_param[12-13]	*open_ofun*	Open function.	
smb_param[14-17]	*open_size*	Bytes to reserve on create or truncate. This field is advisory only.	
smb_param[18-21]	*open_rsvd*[5]	Reserved. Must be zero.	
smb_param[22-23]	*open_pathname*[ ]	File pathname.	

smb_data[ ] FEALIST structure for the file opened.

Secondary Request Format

There may be zero or more of these.

smb_wct Value = 9.

smb_tpscnt Total number of parameter bytes being sent.

smb_tdscnt Total number of data bytes being sent.

smb_pscnt Value = 0. All parameters were in the primary request.

smb_psoff Value = 0. No parameters in secondary request.

smb_psdisp Value = 0. No parameters in secondary request.

smb_dscnt Number of data bytes being sent in this buffer.

smb_dsoff Offset from the start of an SMB header to the data bytes.

smb_dsdisp	Byte displacement for these data bytes.
smb_fid	Value = 0xffff. No FID in this request.
smb_bcc	Total bytes following including pad bytes.
smb_data[]	Data bytes.

Response Format

smb_wct	Value = 10.
smb_tprcnt	Total parameter length returned.
smb_tdrcnt	Value = 0. No data bytes.
smb_prcnt	Number of parameter bytes returned in this buffer.
smb_proff	Offset from the start of an SMB header to the parameter bytes.
smb_prdisp	Value = 0. Byte displacement for these parameter bytes
smb_drcnt	Value = 0. No data bytes.
smb_droff	Value = 0. No data bytes
smb_drdisp	Value = 0. No data bytes
smb_suwcnt	Value = 0. No setup return fields.
smb_bcc	Total bytes following including pad bytes.
smb_param[]	The parameter block for the the *TRANSACT2_OPEN* function response is the open-specific return information in the following format:

Location	Name	Meaning
smb_param[0-1]	*open_fid*	FID.
smb_param[2-3]	+*open_attribute*	Attributes of file or device. Reference Section 5.3.3 on page 43.
smb_param[4-7]	+*open_time*	Last modification time. Reference Section 5.3.1 on page 43.
smb_param[8-11]	+*open_size*	32-bit integer specifying the current file size.
smb_param[12-13]	+*open_access*	Access permissions actually allowed. Reference Section 5.3.7 on page 46.
smb_param[14-15]	+*open_type*	File type. Reference Section 5.3.6 on page 45.
smb_param[16-17]	+*open_state*	State of IPC device (for example, named pipe). Reference X/Open CAE Specification, IPC Mechanisms for SMB.
		Bit 15 — Blocking. Zero (0) indicates that reads/writes block if no data is available; 1 indicates that reads/writes return immediately if no data is available.
		Bit 14 — Endpoint. Zero (0) indicates SMB redirector end of a named pipe; 1 indicates the LMX server end of a named pipe.
		Bits 10-11 — Type of named pipe. 00 indicates the named pipe is a stream mode pipe; 01 indicates the named pipe is a message mode pipe.
		Bits 8-9 — Read Mode. 00 indicates to read the named pipe as a stream mode named pipe; 01 indicates to read the named pipe as a message mode named pipe.
smb_param[18-19]	*open_action*	Action taken.
		Bit 15 — Lock Status. Set true only if an opportunistic lock was requested by the SMB redirector and was granted by the LMX server. This bit should be false (0) if no lock was requested, the

Location	Name	Meaning	
			lock could not be granted, or the LMX server does not support opportunistic locking.
		Bits 0-1	Open Action. The LMX server should set this to match the requested action from the *smb_ofun* field: 1 The file existed and was opened. 2 The file did not exist and was therefore created. 3 The file existed and was truncated.
smb_param[20-23]	*open_fileid*		A unique number for this instance of the file. Similar to a file node number. This value is informational only. If the LMX server does not support the value it may be set to zero.
smb_param[24-25]	*open_offerror*		16-bit integer offset into FEALIST data of first error which occurred while setting the extended attributes.
smb_param[12-13]	++*open_EAlength*		16-bit integer specifying the total EA length for the opened file.

Where:

\+ items returned only if bit 0 of *open_flags2* is set in primary request

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

16.3 TRANSACT2_FINDFIRST

The function code *TRANSACT2_FINDFIRST* in *smb_setup*[0] in the primary *SMBtrans2* request identifies a request to find the first file that matches the specified file specification.

Primary Request Format

smb_wct	Value = 15.
smb_tpscnt	Total number of parameter bytes being sent.
smb_tdscnt	Total number of data bytes being sent.
smb_mprcnt	Maximum return parameter length.
smb_mdrcnt	Maximum return data length.
smb_msrcnt	Value = 0. No setup fields to return.
smb_flags	Bit 0 and bit 1 must be zero.
smb_timeout	Value = 0. Not used for find first.
smb_rsvd1	Reserved. Must be zero.
smb_pscnt	Value = *smb_tpscnt*. All parameters must be in primary request.
smb_psoff	Offset from the start of an SMB header to the parameter bytes.
smb_dscnt	Number of data bytes being sent in this buffer.
smb_dsoff	Offset from the start of an SMB header to the data bytes.
smb_suwcnt	Value = 1.
smb_setup[0]	Value = *TRANSACT2_FINDFIRST*.
smb_bcc	Total bytes following including pad bytes.
smb_param[]	The parameter block for the *TRANSACT2_FINDFIRST* function is the find first-specific information in the following format:

Location	Name	Meaning
smb_param[0-1]	*findfirst_Attribute*	Search attribute.
smb_param[2-3]	*findfirst_SearchCount*	Number of entries to find.
smb_param[3-4]	*findfirst_flags*	Find flags:
		Bit 0 If set, close search after this request.
		Bit 1 If set, close search if end of search reached.
		Bit 2 If set, the SMB redirector requires resume key for each entry found.
smb_param[5-6]	*findfirst_FileInfoLevel*	Search level.
smb_param[7-10]	*findfirst_rsvd*	Reserved. Must be zero.
smb_param[11]	*findfirst_FileName*[]	Beginning of name of the file to find.
smb_param[]	*smb_data*[]	Additional FileInfoLevel-dependent match information. For a search requiring extended attribute matching the data buffer contains the FEALIST data for the search. This location follows after the *findfirst_FileName* field.

Secondary Request Format

There may be zero or more of these.

smb_wct	Value = 9.
smb_tpscnt	Total number of parameter bytes being sent.
smb_tdscnt	Total number of data bytes being sent.
smb_pscnt	Value = 0. All parameters in primary request.
smb_psoff	Value = 0. No parameters in secondary request.
smb_psdisp	Value = 0. No parameters in secondary request.
smb_dscnt	Number of data bytes being sent in this buffer.
smb_dsoff	Offset from the start of an SMB header to the data bytes.
smb_dsdisp	Byte displacement for these data bytes.
smb_fid	Value = 0xffff. No FID in this request.
smb_bcc	Total bytes following including pad bytes.
smb_fid	Value = 0xffff. No FID in this request.
smb_data[]	Data bytes (size = value of *smb_dscnt*).

First Response Format

smb_wct Value = 10.

smb_tprcnt Value = 10.

smb_tdrcnt Total length of return data buffer.

smb_rsvd Reserved. Must be zero.

smb_prcnt Number of parameter bytes returned in this buffer.

smb_proff Offset from the start of an SMB header to the parameter bytes.

smb_prdisp Value = 0. Byte displacement for parameter bytes.

smb_drcnt Number of data bytes returned in this buffer.

smb_droff Offset from the start of an SMB header to the data bytes.

smb_drdisp Byte displacement for these data bytes.

smb_suwcnt Value = 0 No setup return fields.

smb_bcc Total bytes following including pad bytes.

smb_param[] The parameter block for the *TRANSACT2_FINDFIRST* function response is the find first-specific return information in the following format:

Location	Name	Meaning
smb_param[0]	*findfirst_dir_handle*	Directory search handle.
smb_param[0]	*findfirst_searchcount*	Number of matching entries found.
smb_param[0]	*findfirst_eos*	End of search indicator.
smb_param[0]	*findfirst_offerror*	Error offset if EA error.
smb_param[0]	*findfirst_lastname*	If zero, the LMX server does not require *findnext_FileName*[] in order to continue search. If not zero, offset from start of returned data to filename of last found entry returned.

smb_data[] Return data bytes (size = value of *smb_dscnt*). The data block contains the level-dependent information about the matches found in the search. If bit 2 in the *findfirst_flags* is set, each returned file descriptor block will be proceeded by a four-byte resume key.

Subsequent Response Format

smb_wct Value = 10.

smb_tprcnt Value = 8.

smb_tdrcnt Total length of return data buffer.

smb_prcnt Value = 0.

smb_proff Value = 0.

smb_prdisp Value = 0.

smb_drcnt Number of data bytes returned in this buffer.

smb_droff Offset from the start of an SMB header to the data bytes.

smb_drdisp	Byte displacement for these data bytes.
smb_suwcnt	Value = 0. No setup return fields.
smb_bcc	Total bytes following including pad bytes.
smb_data[]	Return data bytes (size = *smb_dscnt*). The data block contains the level-dependent information about the matches found in the search. If bit 2 in the *findfirst_flags* is set, each returned file descriptor block will be proceeded by a four-byte resume key.

16.4 TRANSACT2_FINDNEXT

The function code *TRANSACT2_FINDNEXT* in *smb_setup*[0] in the primary *SMBtrans2* request identifies a request to continue a file search started by a *TRANSACT2_FINDFIRST* search.

Primary Request Format

smb_wct	Value = 15.
smb_tpscnt	Total number of parameter bytes being sent.
smb_tdscnt	Total number of data bytes being sent.
smb_mprcnt	Maximum return parameter length.
smb_mdrcnt	Maximum return data length.
smb_msrcnt	Value = 0. No setup fields to return.
smb_flags	Bit 0 and bit 1 must be zero.
smb_timeout	Value = 0. Not used for find next.
smb_rsvd1	Reserved. Must be zero.
smb_pscnt	Value = *smb_tpscnt*. All parameters must be in primary request.
smb_psoff	Offset from the start of an SMB header to the parameter bytes.
smb_dscnt	Number of data bytes being sent in this buffer.
smb_dsoff	Offset from the start of an SMB header to the data bytes.
smb_suwcnt	Value = 1.
smb_setup[0]	Value = *TRANSACT2_FINDNEXT*.
smb_bcc	Total bytes following including pad bytes.
smb_param[]	The parameter block for the *TRANSACT2_FINDNEXT* function is the find next-specific information in the following format:

Location	Name	Meaning
smb_param[1-2]	*findnext_DirHandle*	Directory search handle.
smb_param[3-4]	*findnext_SearchCount*	Number of entries to find.
smb_param[5-6]	*findnext_FileInfoLevel*	Search level.
smb_param[7-10]	*findnext_ResumeKey*	Server reserved resume key.
smb_param[11-12]	*findnext_flags*	Find flags:
		Bit 0 If set, close search after this request.
		Bit 1 If set, close search if end of search reached.
		Bit 2 If set, the SMB redirector requires resume key for each entry found. If clear, rewind after search.
smb_param[13]	*findnext_FileName*[]	Beginning of name of file to resume search.
smb_param[]	*smb_data*[]	Additional FileInfoLevel-dependent match information. For a search requiring extended attribute matching the data buffer contains the FEALIST data for the seach.

Secondary Request Format

There may be zero or more of these.

smb_wct	Value = 9.
smb_tpscnt	Total number of parameter bytes being sent.
smb_tdscnt	Total number of data bytes being sent.
smb_pscnt	Value = 0. All parameters in primary request.
smb_psoff	Value = 0. No parameters in secondary request.
smb_psdisp	Value = 0. No parameters in secondary request.
smb_dscnt	Number of data bytes being sent in this buffer.
smb_dsoff	Offset from the start of an SMB header to the data bytes.
smb_dsdisp	Byte displacement for these data bytes.
smb_fid	Search handle returned from *TRANSACT2_FINDFIRST*.
smb_bcc	Total bytes following including pad bytes.
smb_data[]	Data bytes (size = *smb_dscnt*).

First Response Format

smb_wct Value = 10.

smb_tprcnt Value = 6.

smb_tdrcnt Total length of return data buffer.

smb_rsvd Reserved. Must be zero.

smb_prcnt Number of parameter bytes returned in this buffer.

smb_proff Offset from the start of an SMB header to the parameter bytes.

smb_prdisp Value = 0. Byte displacement for parameter bytes.

smb_drcnt Number of data bytes returned in this buffer.

smb_droff Offset from the start of an SMB header to the data bytes.

smb_drdisp Byte displacement for these data bytes.

smb_suwcnt Value = 0. No setup return fields.

smb_bcc Total bytes following including pad bytes.

smb_param[] The parameter block for the *TRANSACT2_FINDNEXT* function response is the find next-specific return information in the following format:

Location	Name	Meaning
smb_param[0]	*findnext_searchcount*	Number of matching entries found.
smb_param[1]	*findnext_eos*	End of search indicator.
smb_param[2]	*findnext_offerror*	Error offset if EA error.
smb_param[3]	*findfirst_lastname*	If zero, LMX server does not require *findnext_FileName*[] in order to continue search. If not zero, offset from start of returned data to filename of last found entry returned.
smb_param[4]	*smb_data*[]	Return data bytes (size = *smb_dscnt*). The data block contains the level-dependent information about the matches found in the search. If bit 2 in the *findfirst_flags* is set, each returned file descriptor block will be proceeded by a four-byte resume key.

Subsequent Respoose Format

smb_wct Value = 10.

smb_tprcnt Value = 6.

smb_tdrcnt Total length of return data buffer.

smb_rsvd Reserved. Must be zero.

smb_prcnt Value = 0.

smb_proff Value = 0.

smb_prdisp Value = 0.

smb_drcnt	Number of data bytes returned in this buffer.
smb_droff	Offset from the start of an SMB header to the data bytes.
smb_drdisp	Byte displacement for these data bytes.
smb_suwcnt	Value = 0. No setup return fields.
smb_bcc	Total bytes following including pad bytes.
smb_data[]	Return data bytes (size = *smb_dscnt*). The data block contains the level-dependent information about the matches found in the search. If bit 2 in the *findfirst_flags* is set, each returned file descriptor block will be proceeded by a four-byte resume key.

16.5 TRANSACT2_QFSINFO

The function code *TRANSACT2_QFSINFO* in *smb_setup*[0] in the primary *SMBtrans2* requests identifies a request to query information about a file system.

Primary Request Format

smb_wct	Value = 15.
smb_tpscnt	Total number of parameter bytes being sent.
smb_tdscnt	Total number of data bytes being sent.
smb_mprcnt	Maximum return parameter length.
smb_mdrcnt	Maximum return data length.
smb_msrcnt	Value = 0. No setup fields to return.
smb_flags	Bit 0 and bit 1 must be zero.
smb_timeout	Value = 0. Not used for *SMBtrans2(TRANSACT2_QFSINFO)*.
smb_rsvd1	Reserved. Must be zero.
smb_pscnt	Value = 2. Parameters are in primary request.
smb_psoff	Offset from the start of an SMB header to the parameter bytes.
smb_dscnt	Value = 0. No data sent with *SMBtrans2(TRANSACT2_QFSINFO)*.
smb_dsoff	Value = 0. No data sent with qfsinfo.
smb_suwcnt	Value = 1.
smb_setup[0]	Value = *TRANSACT2_QFSINFO*.
smb_bcc	Total bytes following including pad bytes.
smb_param[]	The parameter block for the *TRANSACT2_QFSINFO* function is the qfsinfo-specific information in the following format:

Location	Name	Meaning
smb_param[0-1]	*qfsinfo_FSInfoLevel*	Level of information required. Refer to DosQFileInfo in the Microsoft OS/2 Programmer's Reference, Volume 4.

Response Format

smb_wct	Value = 10.
smb_tprcnt	Value = 0.
smb_tdrcnt	Total length of return data buffer.
smb_rsvd	Reserved. Must be zero.
smb_prcnt	Value = 0. No return parameter bytes for *TRANSACT2_QFSINFO*.
smb_proff	Offset from the start of an SMB header to the parameter bytes.
smb_prdisp	Value = 0. Byte displacement for parameter bytes.
smb_drcnt	Number of data bytes returned in this buffer.

smb_droff	Offset from the start of an SMB header to the data bytes.
smb_drdisp	Byte displacement for these data bytes.
smb_suwcnt	Value = 0. No setup return fields.
smb_bcc	Total bytes following including pad bytes.
smb_data[]	Return data bytes (size = *smb_dscnt*). The data block contains the level-dependent information about the file system.

16.6 TRANSACT2_SETFSINFO

The function code *TRANSACT2_SETFSINFO* in *smb_setup*[0] in the primary *SMBtrans2* requests identifies a request to set information for a file system subtree.

Primary Request Format

smb_wct	Value = 15.
smb_tpscnt	Total number of parameter bytes being sent.
smb_tdscnt	Total number of data bytes being sent.
smb_mprcnt	Maximum return parameter length.
smb_mdrcnt	Value = 0. No data returned.
smb_msrcnt	Value = 0. No setup fields to return.
smb_flags	Bit 0 and bit 1 must be zero.
smb_timeout	Value = 0. Not used for setfsinfo.
smb_rsvd1	Reserved. Must be zero.
smb_pscnt	Value = 4. All parameters must be in primary request.
smb_psoff	Offset from the start of an SMB header to the parameter bytes.
smb_dscnt	Number of data bytes being sent in this buffer.
smb_dsoff	Offset from the start of an SMB header to the data bytes.
smb_suwcnt	Value = 1.
smb_setup[0]	Value = *TRANSACT2_SETFSINFO*.
smb_bcc	Total bytes following including pad bytes.
smb_param[]	The parameter block for the *TRANSACT2_SETFSINFO* function is the setfsinfo-specific information in the following format:

Location	Name	Meaning
smb_param[0..1]	*setfsinfo_FSInfoLevel*	Level of information provided. Refer to DosQFileInfo in the Microsoft OS/2 Programmer's Reference, Volume 4.

smb_data[]	Level-dependent file system information.

Secondary Request Format

There may be zero or more of these.

smb_wct	Value = 9.
smb_tpscnt	Total number of parameter bytes being sent.
smb_tdscnt	Total number of data bytes being sent.
smb_pscnt	Value = 0. All parameters in primary request.
smb_psoff	Value = 0. No parameters in secondary request.
smb_psdisp	Value = 0. No parameters in secondary request.

smb_dscnt	Number of data bytes being sent in this buffer.
smb_dsoff	Offset from the start of an SMB header to the data bytes.
smb_dsdisp	Byte displacement for these data bytes.
smb_fid	Value = 0xffff. No FID in request.
smb_bcc	Total bytes following including pad bytes.
smb_data[]	Data bytes (size = *smb_dscnt*).

Response Format

smb_wct	Value = 10.
smb_tprcnt	Value = 0.
smb_tdrcnt	Value = 0. No data bytes.
smb_rsvd	Reserved. Must be zero.
smb_prcnt	Value = 0. No return parameters for setfsinfo.
smb_proff	Offset from the start of an SMB header to the parameter bytes.
smb_prdisp	Value = 0. Byte displacement for parameter bytes.
smb_drcnt	Value = 0. No data bytes.
smb_droff	Value = 0. No data bytes.
smb_drdisp	Value = 0. No data bytes.
smb_suwcnt	Value = 0. No setup return fields.
smb_bcc	Value = 0.

16.7 TRANSACT2_QPATHINFO

The function code *TRANSACT2_QPATHINFO* in *smb_setup*[0] in the primary *SMBtrans2* requests identifies a request to query information about specific file or subdirectory.

Primary Request Format

smb_wct Value = 15.

smb_tpscnt Total number of parameter bytes being sent.

smb_tdscnt Total number of data bytes being sent.

smb_mprcnt Maximum return parameter length.

smb_mdrcnt Maximum return data length.

smb_msrcnt Value = 0. No setup fields to return.

smb_flags Bit 0 and bit 1 must be zero.

smb_timeout Value = 0. Not used for qpathinfo.

smb_rsvd1 Reserved. Must be zero.

smb_pscnt Value = *smb_tpscnt*. All parameters must be in primary request.

smb_psoff Offset from the start of an SMB header to the parameter bytes.

smb_dscnt Number of data bytes being sent in this buffer.

smb_dsoff Offset from the start of an SMB header to the data bytes.

smb_suwcnt Value = 1.

smb_setup[0] Value = *TRANSACT2_QPATHINFO*.

smb_bcc Total bytes following including pad bytes.

smb_param[] The parameter block for the *TRANSACT2_QPATHINFO* function is the qpathinfo-specific information in the following format:

Location	Name	Meaning
smb_param[0,1]	*qpathinfo_FSInfoLevel*	Level of information required. Refer to DosQFileInfo in the Microsoft OS/2 Programmer's Reference, Volume 4.
smb_param[2-5]	*qpathinfo_rsvd*	Reserved. Must be zero.
smb_param[6]	*qpathinfo_PathName[]*	File/directory name.

smb_data[] Additional FileInfoLevel-dependent information.

Secondary Request Format

There may be zero or more of these.

smb_wct Value = 9.

smb_tpscnt Total number of parameter bytes being sent.

smb_tdscnt Total number of data bytes being sent.

smb_pscnt Value = 0. All parameters in primary request.

smb_psoff Value = 0. No parameters in secondary request.

smb_psdisp Value = 0. No parameters in secondary request.

smb_dscnt Number of data bytes being sent in this buffer.

smb_dsoff Offset from the start of an SMB header to the data bytes.

smb_dsdisp Byte displacement for these data bytes.

smb_fid Value = 0xffff. No FID in request.

smb_bcc Total bytes following including pad bytes.

smb_data[] Data bytes (size = *smb_dscnt*).

First Response Format

smb_wct Value = 10.

smb_tprcnt Value = 2

smb_tdrcnt Total length of return data buffer.

smb_rsvd Reserved. Must be zero.

smb_prcnt Value = 2. Parameter bytes returned for *TRANSACT2_QFSINFO*.

smb_proff Offset from the start of an SMB header to the parameter bytes.

smb_prdisp Value = 0. Byte displacement for parameter bytes.

smb_drcnt Number of data bytes returned in this buffer.

smb_droff Offset from the start of an SMB header to the data bytes.

smb_drdisp Byte displacement for these data bytes.

smb_suwcnt Value = 0. No set up return fields.

smb_bcc Total bytes following including pad bytes.

smb_param[] The parameter block for the *TRANSACT2_QPATHINFO* response is the qpathinfo-specific return information in the following format:

Location	Name	Meaning
smb_param[0-1]	*qpathinfo_offerror*	Error offset if EA error.

smb_data[] Return data bytes (size = *smb_dscnt*). The data block contains the requested level-dependent information about the path.

Subsequent Response Format

smb_wct Value = 10.

smb_tprcnt Value = 2

smb_tdrcnt Total length of return data buffer.

smb_rsvd Reserved. Must be zero.

smb_prcnt Value = 0.

smb_proff Value = 0.

smb_prdisp Value = 0.

smb_drcnt	Number of data bytes returned in this buffer.
smb_droff	Offset from the start of an SMB header to the data bytes.
smb_drdisp	Byte displacement for these data bytes.
smb_suwcnt	Value = 0. No set up return fields.
smb_bcc	Total bytes following including pad bytes.
smb_data[]	Return data bytes (size = *smb_dscnt*). The data block contains the requested level-dependent information about the path.

16.8 TRANSACT2_SETPATHINFO

The function code *TRANSACT2_SETPATHINFO* in *smb_setup*[0] in the primary *SMBtrans2* requests identifies a request to set information for a file or directory.

Primary Request Format

smb_wct Value = 15.

smb_tpscnt Total number of parameter bytes being sent.

smb_tdscnt Total number of data bytes being sent.

smb_mprcnt Maximum return parameter length.

smb_mdrcnt Value = 0. No data returned.

smb_msrcnt Value = 0. No setup fields to return.

smb_flags Bit 0 and bit 1 must be zero.

smb_timeout Value = 0. Not used for setpathinfo.

smb_rsvd1 Reserved. Must be zero.

smb_pscnt Value = *smb_tpscnt*. All parameters must be in primary request.

smb_psoff Offset from the start of an SMB header to the parameter bytes.

smb_dscnt Number of data bytes being sent in this buffer.

smb_dsoff Offset from the start of an SMB header to the data bytes.

smb_suwcnt Value = 1.

smb_setup[0] Value = *TRANSACT2_SETPATHINFO*.

smb_bcc Total bytes following including pad bytes.

smb_param[] The parameter block for the *TRANSACT2_SETPATHINFO* function is the setpathinfo-specific information in the following format:

Location	Name	Meaning
smb_param[0-1]	*setpathinfo_PathInfoLevel*	Information level supplied.
smb_param[2-5]	*setpathinfo_rsvd*	Reserved. Must be zero.
smb_param[6]	*setpathinfo_pathname[]*	Pathname to set information on.

smb_data[] Additional FileInfoLevel-dependent information.

Secondary Request Format

There may be zero or more of these.

smb_wct Value = 9.

smb_tpscnt Total number of parameter bytes being sent.

smb_tdscnt Total number of data bytes being sent.

smb_pscnt Value = 0. All parameters in primary request.

smb_psoff Value = 0. No parameters in secondary request.

smb_psdisp Value = 0. No parameters in secondary request.

smb_dscnt	Number of data bytes being sent in this buffer.
smb_dsoff	Offset from the start of an SMB header to the data bytes.
smb_dsdisp	Byte displacement for these data bytes.
smb_fid	Value = 0xffff. No FID in this request.
smb_bcc	Total bytes following including pad bytes.
smb_data[]	Data bytes (size = *smb_dscnt*).

Response Format

smb_wct	Value = 10.
smb_tprcnt	Value = 2.
smb_tdrcnt	Value = 0. No data bytes.
smb_rsvd	Reserved. Must be zero.
smb_prcnt	Value = 2. Parameter bytes being returned.
smb_proff	Offset from the start of an SMB header to the parameter bytes.
smb_prdisp	Value = 0. Byte displacement for parameter bytes.
smb_drcnt	Value = 0. No data bytes.
smb_droff	Value = 0. No data bytes.
smb_drdisp	Value = 0. No data bytes.
smb_suwcnt	Value = 0. No set up return fields.
smb_bcc	Total bytes following including pad bytes.
smb_param[]	The parameter block for the *TRANSACT2_SETPATHINFO* function response is the setpathinfo-specific return information in the following format:

Location	Name	Meaning
smb_param[0-1]	*setpathinfo_offerror*	Offset into FEALIST data of first error which occurred while setting the extended attributes.

16.9 TRANSACT2_QFILEINFO

The function code *TRANSACT2_QFILEINFO* in *smb_setup*[0] in the primary *SMBtrans2* requests identifies a request to query information about a specific file.

Primary Request Format

smb_wct Value = 15.

smb_tpscnt Total number of parameter bytes being sent.

smb_tdscnt Total number of data bytes being sent.

smb_mprcnt Maximum return parameter length.

smb_mdrcnt Maximum return data length.

smb_msrcnt Value = 0. No setup fields to return.

smb_flags Bit 0 and bit 1 must be zero.

smb_timeout Value = 0. Not used for qfileinfo.

smb_rsvd1 Reserved. Must be zero.

smb_pscnt Value = 4. All parameters are in primary request.

smb_psoff Offset from the start of an SMB header to the parameter bytes.

smb_dscnt Number of data bytes being sent in this buffer.

smb_dsoff Offset from the start of an SMB header to the data bytes.

smb_suwcnt Value = 1.

smb_setup[0] Value = *TRANSACT2_QFILEINFO*.

smb_bcc Total bytes following including pad bytes.

smb_param[] The parameter block for the *TRANSACT2_QFILEINFO* function is the qfileinfo-specific information in the following format:

Location	Name	Meaning
smb_param[0-1]	*qfileinfo_FileHandle*	FID.
smb_param[2-3]	*qfileinfo_FileInfoLevel*	Level of information required. Refer to DosQFileInfo in the Microsoft OS/2 Programmer's Reference, Volume 4.

smb_data[] Additional FileInfoLevel-dependent information.

Secondary Request Format

There may be zero or more of these.

smb_wct Value = 9.

smb_tpscnt Total number of parameter bytes being sent.

smb_tdscnt Total number of data bytes being sent.

smb_pscnt Value = 0.

smb_psoff Value = 0.

smb_psdisp Value = 0.

smb_dscnt Number of data bytes being sent in this buffer.

smb_dsoff Offset from the start of an SMB header to the data bytes.

smb_dsdisp Byte displacement for these data bytes.

smb_fid The FID.

smb_bcc Total bytes following including pad bytes.

smb_data[] Data bytes (size = *smb_dscnt*).

First Response Format

smb_wct Value = 10.

smb_tprcnt Value = 2.

smb_tdrcnt Total length of return data buffer.

smb_rsvd Reserved. Must be zero.

smb_prcnt Value = 2. No parameter bytes returned for qfileinfo.

smb_proff Offset from the start of an SMB header to the parameter bytes.

smb_prdisp Value = 0. Byte displacement for these parameter bytes.

smb_drcnt Number of data bytes returned in this buffer.

smb_droff Offset from the start of an SMB header to the data bytes.

smb_drdisp Byte displacement for these data bytes.

smb_suwcnt Value = 0. No set up return fields.

smb_bcc Total bytes following including pad bytes.

smb_param[] The parameter block for the *TRANSACT2_QFILEINFO* response is the qfileinfo-specific return information in the following format:

Location	Name	Meaning
smb_param[0-1]	*qfileinfo_offerror*	Error offset if EA error.

smb_data[] Return data bytes (size = *smb_dscnt*). The data block contains the requested level-dependent information about the file.

Subsequent Response Format

smb_wct Value = 10.

smb_tprcnt Value = 2.

smb_tdrcnt Total length of return data buffer.

smb_rsvd Reserved. Must be zero.

smb_prcnt Value = 0.

smb_proff Value = 0.

smb_prdisp Value = 0.

smb_drcnt Number of data bytes returned in this buffer.

smb_droff	Offset from the start of an SMB header to the data bytes.
smb_drdisp	Byte displacement for these data bytes.
smb_suwcnt	Value = 0. No set up return fields.
smb_bcc	Total bytes following including pad bytes.
smb_data[]	Return data bytes (size = *smb_dscnt*). The data block contains the requested level-dependent information about the file.

16.10 TRANSACT2_SETFILEINFO

The function code *TRANSACT2_SETFILEINFO* in *smb_setup*[0] in the primary *SMBtrans2* requests identifies a request to set information for a specific file.

Primary Request Format

smb_wct	Value = 15.
smb_tpscnt	Total number of parameter bytes being sent.
smb_tdscnt	Total number of data bytes being sent.
smb_mprcnt	Maximum return parameter length.
smb_mdrcnt	Value = 0. No data returned.
smb_msrcnt	Value = 0. No setup fields to return.
smb_flags	Bit 0 and bit 1 must be zero.
smb_timeout	Value = 0. Not used for setfileinfo.
smb_rsvd1	Reserved. Must be zero.
smb_pscnt	Value = 6. Parameters must be in primary request.
smb_psoff	Offset from the start of an SMB header to the parameter bytes.
smb_dscnt	Number of data bytes being sent in this buffer.
smb_dsoff	Offset from the start of an SMB header to the data bytes.
smb_suwcnt	Value = 1.
smb_setup[0]	Value = *TRANSACT2_SETFILEINFO*.
smb_bcc	Total bytes following including pad bytes.
smb_param[]	The parameter block for the *TRANSACT2_SETFILEINFO* function is the setfileinfo-specific information in the following format:

Location	Name	Meaning
smb_param[0-1]	*setfileinfo_FileHandle*	FID.
smb_param[2-3]	*setfileinfo_FileInfoLevel*	Level of information required. Refer to DosQFileInfo in the Microsoft OS/2 Programmer's Reference, Volume 4.
smb_param[4-5]	*setfileinfo_IOFlag*	Flag field: 0x0010 Write through. 0x0020 No cache.

smb_data[]	Additional FileInfoLevel-dependent information. For level = 2, *smb_data*[] contains the FEALIST structure to set for this file.

Secondary Request Format

There may be zero or more of these.

smb_wct	Value = 9
smb_tpscnt	Total number of parameter bytes being sent.
smb_tdscnt	Total number of data bytes being sent.
smb_pscnt	Value = 0
smb_psoff	Value = 0. No parameters in secondary request.
smb_psdisp	Value = 0
smb_dscnt	Number of data bytes being sent in this buffer.
smb_dsoff	Offset from the start of an SMB header to the data bytes.
smb_dsdisp	Byte displacement for these data bytes.
smb_fid	The FID.
smb_bcc	Total bytes following including pad bytes.
smb_data[]	Data bytes (size = *smb_dscnt*).

Response Format

smb_wct	Value = 10
smb_tprcnt	Value = 2
smb_tdrcnt	Value = 0. No data bytes.
smb_rsvd	Reserved. Must be zero.
smb_prcnt	Value = 2. Parameter bytes being returned.
smb_proff	Offset from the start of an SMB header to the parameter bytes.
smb_prdisp	Value = 0. Byte displacement for these parameter bytes.
smb_drcnt	Value = 0. No data bytes.
smb_droff	Value = 0. No data bytes.
smb_drdisp	Value = 0. No data bytes.
smb_suwcnt	Value = 0. No set up return fields.
smb_bcc	Total bytes following including pad bytes.
smb_param[]	The parameter block for the *TRANSACT2_SETFILEINFO* function response is the setfileinfo-specific return information in the following format:

Location	Name	Meaning
smb_param[0 1]	*setfileinfo_offerror*	Offset into FEALIST data of first error which occurred while setting the extended attributes.

16.11 TRANSACT2_FINDNOTIFYFIRST

The function code *TRANSACT2_FINDNOTIFYFIRST* in *smb_setup*[0] in the primary *SMBtrans2* request identifies a request to commence monitoring changes to a specific file or directory.

Primary Request Format

smb_wct Value = 15.

smb_tpscnt Total number of parameter bytes being sent.

smb_tdscnt Total number of data bytes being sent.

smb_mprcnt Maximum return parameter length.

smb_mdrcnt Maximum return data length.

smb_msrcnt Value = 0. No setup fields to return.

smb_flags Bit 0 and bit 1 must be zero.

smb_timeout Specifies duration to wait for changes.

smb_rsvd1 Reserved. Must be zero.

smb_pscnt Value = tpscnt. All parameters must be in primary request.

smb_psoff Offset from the start of an SMB header to the parameter bytes.

smb_dscnt Number of data bytes being sent in this buffer.

smb_dsoff Offset from the start of an SMB header to the data bytes.

smb_suwcnt Value = 1.

smb_setup[0] Value = *TRANSACT2_FINDNOTIFYFIRST*.

smb_bcc Total bytes following including pad bytes.

smb_param[] The parameter block for the *TRANSACT2_FINDNOTIFYFIRST* function is the find first-specific information in the following format:

Location	Name	Meaning
smb_param[0-1]	*findnfirst_Attribute*	Search attribute.
smb_param[2-3]	*findnfirst_ChangeCount*	Number of changes to wait for.
smb_param[4-5]	*findnfirst_Level*	Information level required.
smb_param[6-9]	*findfirst_rsvd*	Reserved. Must be zero.
smb_param[10]	*findnfirst_PathSpec[]*	Path to monitor.

smb_data[] Additional level-dependent match data.

Secondary Request Format

There may be zero or more of these.

smb_wct Value = 9.

smb_tpscnt Total number of parameter bytes being sent.

smb_tdscnt Total number of data bytes being sent.

smb_pscnt Value = 0. All parameters in primary request.

smb_psoff Value = 0. No parameters in secondary request.

smb_psdisp Value = 0. No parameters in secondary request.

smb_dscnt Number of data bytes being sent in this buffer.

smb_dsoff Offset from the start of an SMB header to the data bytes.

smb_dsdisp Byte displacement for these data bytes.

smb_fid Value = 0xffff. No FID in this request.

smb_bcc Total bytes following including pad bytes.

smb_data[] Data bytes (size = *smb_dscnt*).

First Response Format

smb_wct Value = 10.

smb_tprcnt Value = 6.

smb_tdrcnt Total length of return data buffer.

smb_rsvd Reserved. Must be zero.

smb_prcnt Number of parameter bytes returned in this buffer.

smb_proff Offset from the start of an SMB header to the parameter bytes.

smb_prdisp Value = 0. Byte displacement for these parameter bytes.

smb_drcnt Number of data bytes returned in this buffer.

smb_droff Offset from the start of an SMB header to the data bytes.

smb_drdisp Byte displacement for these data bytes.

smb_suwcnt Value = 0. No set up return fields.

smb_bcc Total bytes following including pad bytes.

smb_param[] The parameter block for the *TRANSACT2_FINDNOTIFYFIRST* function response is the find first-specific return information in the following format:

Location	Name	Meaning
smb_param[0-1]	*findnfirst_handle*	Monitor handle.
smb_param[2-3]	*findnfirst_changecount*	Number of changes which occurred within timeout.
smb_param[4-5]	*findnfirst_offerror*	Error offset if EA error.

smb_data[] Data bytes (size = *smb_dscnt*). The data block contains the level-dependent information about the changes which occurred .

Subsequent Response Format

smb_wct Value = 10.

smb_tprcnt Value = 6.

smb_tdrcnt Total length of return data buffer.

smb_rsvd Reserved. Must be zero.

smb_prcnt Value = 0.

smb_proff Value = 0.

smb_prdisp	Value = 0.
smb_drcnt	Number of data bytes returned in this buffer.
smb_droff	Offset from the start of an SMB header to the data bytes.
smb_drdisp	Byte displacement for these data bytes.
smb_suwcnt	Value = 0. No set up return fields.
smb_bcc	Total bytes following including pad bytes.
smb_data[]	Data bytes (size = *smb_dscnt*). The data block contains the level-dependent information about the changes which occurred.

16.12 TRANSACT2_FINDNOTIFYNEXT

The function code *TRANSACT2_FINDNOTIFYNEXT* in *smb_setup*[0] in the primary *SMBtrans2* request identifies a request to continue monitoring changes to a file or directory specified by a *TRANSACT_FINDNOTIFYFIRST* request.

Primary Request Format

smb_wct	Value = 15.
smb_tpscnt	Total number of parameter bytes being sent.
smb_tdscnt	Total number of data bytes being sent.
smb_mprcnt	Maximum return parameter length.
smb_mdrcnt	Maximum return data length.
smb_msrcnt	Value = 0. No setup fields to return.
smb_flags	Bit 0 and bit 1 must be zero.
smb_timeout	Duration of monitor period.
smb_rsvd1	Reserved. Must be zero.
smb_pscnt	Value = 0. All parameters in primary request.
smb_psoff	Offset from the start of an SMB header to the parameter bytes.
smb_dscnt	Number of data bytes being sent in this buffer.
smb_dsoff	Offset from the start of an SMB header to the data bytes.
smb_suwcnt	Value = 1.
smb_setup[0]	Value = *TRANSACT2_FINDNOTIFYNEXT*.
smb_bcc	Total bytes following including pad bytes.
smb_param[]	The parameter block for the *TRANSACT2_FINDNOTIFYNEXT* function is the find next-specific information in the following format:

Location	Name	Meaning
smb_param[0-1]	*findnnext_DirHandle*	Directory monitor handle.
smb_param[2-3]	*findnnext_ChangeCount*	Number of changes to wait for.

smb_data[]	Data bytes (size = *smb_dscnt*). Additional level-dependent monitor information.

Secondary Request Format

There may be zero or more of these.

smb_wct	Value = 9.
smb_tpscnt	Total number of parameter bytes being sent.
smb_tdscnt	Total number of data bytes being sent.
smb_pscnt	Value = 0. All parameters in primary request.
smb_psoff	Value = 0. No parameters in secondary request.
smb_psdisp	Value = 0. No parameters in secondary request.

smb_dscnt Number of data bytes being sent in this buffer.

smb_dsoff Offset from the start of an SMB header to the data bytes.

smb_dsdisp Byte displacement for these data bytes.

smb_fid Search handle.

smb_bcc Total bytes following including pad bytes.

smb_data[] Data bytes (size = *smb_dscnt*).

First Response Format

smb_wct Value = 10.

smb_tprcnt Value = 4.

smb_tdrcnt Total length of return data buffer.

smb_rsvd Reserved. Must be zero.

smb_prcnt Number of parameter bytes returned in this buffer.

smb_proff Offset from the start of an SMB header to the parameter bytes.

smb_prdisp Value = 0. Byte displacement for these parameter bytes.

smb_drcnt Number of data bytes returned in this buffer.

smb_droff Offset from the start of an SMB header to the data bytes.

smb_drdisp Byte displacement for these data bytes.

smb_suwcnt Value = 0. No set up return fields.

smb_bcc Total bytes following including pad bytes.

smb_param[] The parameter block for the *TRANSACT2_FINDNOTIFYNEXT* function response is the find notify next-specific return information in the following format:

Location	Name	Meaning
smb_param[0-1]	*findnnext_changecount*	Number of changes during the monitor period.
smb_param[2-3]	*findnnext_offerror*	Error offset if EA error.

smb_data[] Data bytes (size = *smb_dscnt*). The data block contains the level-dependent information about the changes which occurred.

Subsequent Response Format

smb_wct Value = 10.

smb_tprcnt Value = 4.

smb_tdrcnt Total length of return data buffer.

smb_rsvd Reserved. Must be zero.

smb_prcnt Value = 0.

smb_proff Value = 0.

smb_prdisp Value = 0.

smb_drcnt	Number of data bytes returned in this buffer.
smb_droff	Offset from the start of an SMB header to the data bytes.
smb_drdisp	Byte displacement for these data bytes.
smb_suwcnt	Value = 0. No set up return fields.
smb_bcc	Total bytes following including pad bytes.
smb_data[]	Data bytes (size = *smb_dscnt*). The data block contains the level-dependent information about the changes which occurred.

16.13 TRANSACT2_MKDIR

The function code *TRANSACT2_MKDIR* in *smb_setup*[0] in the primary *SMBtrans2* requests identifies a request to create a directory with extended attributes.

Primary Request Format

smb_wct Value = 15.

smb_tpscnt Total number of parameter bytes being sent.

smb_tdscnt Total number of data bytes being sent.

smb_mprcnt Maximum return parameter length.

smb_mdrcnt Value = 0. No data returned.

smb_msrcnt Value = 0. No setup fields to return.

smb_flags Bit 0 and bit 1 must be zero.

smb_timeout Value = 0.

smb_rsvd1 Reserved. Must be zero.

smb_pscnt Value = 0. All parameters in primary request.

smb_psoff Offset from the start of an SMB header to the parameter bytes.

smb_dscnt Number of data bytes being sent in this buffer.

smb_dsoff Offset from the start of an SMB header to the data bytes.

smb_suwcnt Value = 1.

smb_setup[0] Value = *TRANSACT2_MKDIR*.

smb_bcc Total bytes following including pad bytes.

smb_param[] The parameter block for the *TRANSACT2_MKDIR* function is the mkdir-specific information in the following format:

Location	Name	Meaning
smb_param[0-3]	*mkdir_rsvd*	Reserved. Must be zero.
smb_param[4]	*mkdir_dirname*[]	Beginning of directory name.

smb_data[] Data bytes (size = *smb_dscnt*). FEALIST structure for the directory to be created.

Secondary Request Format

There may be zero or more of these.

smb_wct Value = 9.

smb_tpscnt Total number of parameter bytes being sent.

smb_tdscnt Total number of data bytes being sent.

smb_pscnt Value = 0. All parameters in primary request.

smb_psoff Value = 0. No parameters in secondary request.

smb_psdisp Value = 0. No parameters in secondary request.

smb_dscnt Number of data bytes being sent in this buffer.

smb_dsoff Offset from the start of an SMB header to the data bytes.

smb_dsdisp Byte displacement for these data bytes.

smb_fid Value = 0xffff. No FID in this request.

smb_bcc Total bytes following including pad bytes.

smb_data[] Data bytes (size = *smb_dscnt*).

Response Format

smb_wct Value = 10.

smb_tprcnt Value = 2.

smb_tdrcnt Value = 0. No data bytes.

smb_rsvd Reserved. Must be zero.

smb_prcnt Value = 2. Parameter bytes being returned.

smb_proff Offset from the start of an SMB header to the parameter bytes.

smb_prdisp Value = 0. Byte displacement for these parameter bytes.

smb_bcc Total bytes following including pad bytes.

smb_param[] The parameter block for the *TRANSACT2_MKDIR* function response is the mkdir-specific return information in the following format:

Location	Name	Meaning
smb_param[0-1]	*mkdir_offerror*	Offset into FEALIST data of first error which occurred while setting the extended attributes.

Appendix A

SMB Transmission Analysis

A.1 Introduction

This appendix describes the mapping between DOS and OS/2 system calls on an SMB redirector, and the associated SMB requests sent from the SMB redirector to an LMX server. The DOS SMB redirector is assumed to be using the core SMB protocols, and the OS/2 SMB redirector is assumed to be using the LAN Manager extended SMB protocols. While an OS/2 SMB redirector will use core SMB requests to communicate with a core LMX server, and a DOS LAN Manager client will use extended SMB requests to communicate with an OS/2 server, these situations will not be considered here.

The mappings given here do not completely describe the behaviour of all SMB redirectors; they do not take into account various optimisations which SMB redirectors may do which will result in behaviour which differs from that described here. In particular, the extended SMB protocol contains a number of facilities which allow a redirector to improve performance. These include: SMB chaining, opportunistic locking, caching and various specialised SMB requests, such as Read Block Multiplex, Write Block Multiplex, Read Block Raw and Write Block Raw. Redirectors which make use of these facilities may not behave exactly as described here.

It should also be noted that the OS/2 SMB redirector and file system make extensive use of internal buffers and heuristics that make it difficult to determine an exact mapping between OS/2 API calls and SMB emissions. The listed API calls give an indication of which SMBs are sent when invoked, and where possible, an explanation is given regarding any special circumstances.

DOS and OS/2 system calls which are not listed here will not normally result in SMB requests being transmitted.

A.2 DOS Functions

Function Number	DOS Function
0x00	Terminate Programme
0x05	Print Character
0x0d	Reset Disk
0x0f	Open File (FCB I/O)
0x10	Close File (FCB I/O)
0x11	Search For First Entry
0x12	Search For Next Entry
0x13	Delete File (FCB I/O)
0x14	Sequential Read (FCB I/O)
0x15	Sequential Write (FCB I/O)
0x16	Create File (FCB I/O)
0x17	Rename File (FCB I/O)
0x1b	Get Default Drive Data
0x1c	Get Drive Data
0x21	Random Read (FCB I/O)
0x22	Random Write (FCB I/O)
0x23	Get File Size (FCB I/O)
0x27	Random Block Read (FCB I/O)
0x28	Random Block Write (FCB I/O)
0x36	Get Disk Free Space
0x39	Create Directory
0x3a	Remove Directory
0x3b	Change Current Directory
0x3c	Create File Handle
0x3d	Open File Handle
0x3e	Close File Handle
0x3f	Read Via File Handle
0x40	Write Via File Handle
0x41	Delete Directory Entry
0x42	Move File Pointer
0x43	Set/Get File Attributes
0x4b	Load and Execute Programme/Load Overlay
0x4c	End Process
0x4e	Find First File
0x4f	Find Next File
0x56	Change Directory Entry
0x57	Set/Get Date/Time of File
0x5a	Create Temporary File Handle
0x5b	Create New File
0x5c	Unlock/Lock File
0x5f	Get Assign List Entry
0x68	Flush Buffer

Change Current Directory

Function number	0x3b.
SMB sent	*SMBchkpth.*
Reason	Change directory.

Change Directory Entry

Function number	0x56
SMB sent	*SMBmv.*
Reason	Rename file.

Close File (FCB I/O)

Function number	0x10
SMB sent	*SMBclose.*
Reason	Close file (FCB I/O).

Close File Handle

Function number	0x3e.
SMB sent	*SMBclose, SMBsplclose* (printer device).
Reason	Close file.

Create Directory

Function number	0x39
SMB sent	*SMBmkdir.*
Reason	Make directory.

Create File (FCB I/O)

Function number	0x16
SMB sent	*SMBcreate.*
Reason	Create file.

Create File Handle

Function number	0x3c.
SMB sent	*SMBcreate.*
Reason	Create file.

Create New File

Function number	0x5b.
SMB sent	*SMBmknew*.
Reason	Create file.

Delete Directory Entry

Function number	0x41.
SMB sent	*SMBunlink*.
Reason	Delete file.

Delete File (FCB I/O)

Function number	0x 13
SMB sent	*SMBunlink*.
Reason	Delete file (FCB I/O).

End Process

Function number	0x4c.
SMB sent	*SMBexit*.
Reason	Exit programme.

Find First File

Function number	0x4e.
SMB sent	*SMBsearch*.
Reason	Find first matching filename.

Find Next File

Function number	0x4f.
SMB sent	*SMBsearch*.
Reason	Find next matching filename.

Flush Buffer

Function number	0x68
SMB sent	*SMBflush*.
Reason	Commit file.

Get Assigo List Entry

Function number	0x5f.
SMB sent	*SMBtcon, SMBtdis.*
Reason	Redirect device, cancel redirection.

Get Default Drive Data

Function number	0x 1b.
SMB sent	*SMBdskattr.*
Reason	Get data on the default drive.

Get Disk Free Space

Function number	0x36
SMB sent	*SMBdskattr.*
Reason	Get free space on disk.

Get Drive data

Function number	0x 1c.
SMB sent	*SMBdskattr.*
Reason	Get data on a drive.

Get File Size (FCB I/O)

Function number	0x23
SMB sent	*SMBsearch.*
Reason	File size in records.

Load and Execute Programme/Load Overlay

Function number	0x4b.
SMB sent	*SMBopen, SMBread, SMBclose.*
Reason	Load/execute programme.

Move File Pointer

Function number	0x42
SMB sent	*SMBlseek.*
Reason	Set position in file.

Open File (FCB I/O)

Function number	0x0f.
SMB sent	*SMBopen* (read/write/share set to 0xff).
Reason	Open file (FCB I/O).

Open File Handle

Function number	0x3d.
SMB sent	*SMBopen*, *SMBsplopen* (printer device).
Reason	Open file.

Print Character

Function number	0x05
SMB sent	*SMBsplopen*, *SMBsplwr*, *SMBsplclose*.
Reason	Printer output.

Random Block Read (FCB I/O)

Function number	0x27.
SMB sent	*SMBread*.
Reason	Random block read (FCB I/O).

Random Block Write (FCB I/O)

Function number	0x28
SMB sent	*SMBwrite*.
Reason	Random block write (FCB I/O).

Random Read (FCB I/O)

Function number	0x21.
SMB sent	*SMBread*.
Reason	Random read (FCB I/O).

Random Write (FCB I/O)

Function number	0x22
SMB sent	*SMBwrite*.
Reason	Random write.

Read Via File Handle

Function number	0x3f.
SMB sent	*SMBread.*
Reason	Read file.

Remove Directory

Function number	0x3a.
SMB sent	*SMBrmdir.*
Reason	Remove directory.

Rename File (FCB I/O)

Function number	0x17.
SMB sent	*SMBmv.*
Reason	Rename file.

Reset Disk

Function number	0x0d.
SMB sent	*SMBflush.*
Reason	Disk reset (flush file buffers).

Search For First Entry

Function number	0x11.
SMB sent	*SMBsearch.*
Reason	Search first matching entry.

Search For Next Entry

Function number	0x12
SMB sent	*SMBsearch.*
Reason	Search next matching entry.

Sequential Read (FCB I/O)

Function number	0x14
SMB sent	*SMBread.*
Reason	Sequential read (FCB I/O).

Sequential Write (FCB I/O)

Function number	0x15
SMB sent	*SMBwrite.*
Reason	Sequential write (FCB I/O).

Set/Get Date/Time of File

Function number	0x57.
SMB sent	*SMBsearch, SMBsetatr.*
Reason	Get/set file date and time.

Set/Get File Attributes

Function number	0x43
SMB sent	*SMBsetatr.*
Reason	Change file attributes.

Terminate Programme

Function number	0x00
SMB sent	*SMBexit.*
Reason	Programme terminate.

Unlock/Lock File

Function number	0x5c.
SMB sent	*SMBlock, SMBunlock.*
Reason	Lock/Unlock file.

Write Via File Handle

Function number	0x40
SMB sent	*SMBwrite, SMBsplwr* (printer device).
Reason	Write file.

A.3 OS/2 Functions

The SMB requests generated from OS/2 redirectors will vary based on the protocol dialect negotiated. This variation is highlighted in the sequences below by listing the SMB request that will be sent if the extended 1.0 dialect was negotiated first followed by the SMB request for the extended 2.0 dialect.

DosBufReset

SMB sent	*SMBflush.*
Reason	Flush file buffer.

DosChDir

SMB sent	*SMBchkpth.*
Reason	Change the current working directory.

DosClose

SMB sent	*SMBclose, SMBwriteclose, SMBwrite.*
Reason	Close FID. If the file I/O is buffered, a *DosClose* will cause the data in the buffers to be flushed. This type of situation may cause an *SMBwriteclose* or *SMBwrite* to be sent.

DosDelete

SMB sent	*SMBunlink.*
Reason	Delete a file.

DosDevIOCtl

SMB sent	*SMBioctl, SMBioctls.*
Reason	Pass a device-specific I/O control request to a driver.

DosExecPgm

SMB sent	*SMBopen, SMBread, SMBclose. SMBtrans2(TRANSACT2_OPEN)* may be used for the open function instead of *SMBopen* for the extended 2.0 dialect.
Reason	Start a programme as a child process. *DosExecPgm* makes use of OS/2s standard file I/O functions.

DosFileLocks

SMB sent	*SMBlock SMBlockingX, SMBlockread, SMBunlock, SMBwriteunlock.*
Reason	Set or reset a byte lock range in an open file. An *SMBwriteunlock* is sent after unlocking bytes which were just written out. *SMBlockread* is used to lock and then read ahead.

DosFindClose

SMB sent *SMBfclose* and possibly *SMBfindnclose*.

Reason Close an active directory search handle. If change notification was involved, the *SMBfindnclose* will be sent to cancel further notifications.

DosFindFirst

SMB sent *SMBffirst* or *SMBtrans2(TRANSACT2_FINDFIRST)*.

Reason Find the first file in a directory matching the search pattern.

DosFindFirst2

SMB sent *SMBtrans2(TRANSACT2_FINDFIRST)*. An *SMBfindclose* may follow.

Reason Find the first file in a directory matching the search pattern. If no additional searchs are desired the *SMBfindclose* will be used to allow the server to free resources associated with the find.

DosFindNext

SMB sent *SMBffirst* or *SMBtrans2(TRANSACT2_FINDNEXT)*.

Reason Get the next file from the search pattern.

If this function is used on a sufficiently large directory it will eventually send an *SMBfind* request.

DosFindNotifyClose

SMB sent *SMBfindnclose*.

Reason To indicate to the LMX server that directory search requests are complete.

DosMkDir

SMB sent *SMBmkdir SMBtrans2(TRANSACT2_MKDIR)*.

Reason Create a new directory.

DosMove

SMB sent *SMBmv*.

Reason Rename or move a file.

DosOpen

SMB sent *SMBopenX, SMBopen, SMBcreate, SMBreadX* or *SMBtrans2(TRANSACT2_OPEN)*.

Reason Open a device/file for I/O.

DosOpen may send an *SMBreadX* read ahead. *DosOpen* will send an *SMBopenX* instead of an *SMBopen* when in protected mode. *SMBopen* has no capabilities for creating a file when opening, so *DosOpen* may send an *SMBcreate*.

DosQCurDir

SMB sent	*SMBchkpth.*
Reason	Determine the current directory of a logical drive.

DosQFSInfo

SMB sent	*SMBdskattr* or *SMBtrans2(TRANSACT2_QFSINFO).*
Reason	Retrieve file system information data.

DosQFileInfo

SMB sent	*SMBgetattrE* or *SMBtrans2(TRANSACT2_QFILEINFO).*
Reason	Retrieve a file information record.

DosQFileMode

SMB sent	*SMBgetatr.*
Reason	Get a file's attribute byte.

DosRead

SMB sent	*SMBread, SMBreadX, SMBreadbraw, SMBreadbmpx.*
Reason	Read characters from an FID.
	SMBreadbraw is used to send a block of data which is larger than the data size which was negotiated.

DosReadAsync

SMB sent	*SMBread, SMBreadX, SMBreadbraw, SMBreadbmpx.*
Reason	Read characters from an FID asynchronously.
	Same behaviour as *DosRead.*

DosRmDir

SMB sent	*SMBrmdir.*
Reason	Delete a subdirectory.

DosSetFileInfo

SMB sent	*SMBsetattrE.*
Reason	Change a file's directory information.

DosSetFileMode

SMB sent	*SMBsetatr.*
Reason	Change a file's attribute.

DosWrite

SMB sent	*SMBwrite, SMBwriteX, SMBwritebraw, SMBwritebmpx.*
Reason	Write characters to an FID.
	SMBwritebraw is used to send a block of data which is larger than the data size which was negotiated.

DosWriteAsync

SMB sent	*SMBwrite, SMBwriteX, SMBwritebraw, SMBwritebmpx.*
Reason	Write characters to an FID asynchronously.
	Same behaviour as *DosWrite.*

Appendix B

LAN Manager Remote Administration Protocol

B.1 Overview

This section describes the mechanism used by LAN Manager to implement remote administration functions and access control lists. The protocols described here are those which are provided by the extended dialects. They are included here so that an implementor can build an LMX server which can handle this class of SMB redirector requests. However, their inclusion in this specification does not imply any X/Open endorsement of these mechanisms as the basis for future X/Open network management functionality.

All administrative functions in the LAN Manager are provided by a set of shared library routines, often referred to as LAN Manager API routines. Many of these routines have a servername argument which the caller uses to distinguish a local administrative operation (one which applies to the LMX server on the local machine) from a remote operation (one which applies to the server on another machine).

In the case of a remote operation the SMB redirector packages up its arguments, and sends them to the appropriate LMX server. The LMX server then calls the corresponding LAN Manager API routine locally, packages the results, and sends them back to the SMB redirector. The mechanism resembles a specialised, private, remote procedure call facility between the SMB redirector and the LMX server.

B.2 Remote API Protocol

1. All remote API operations are done using the share name IPC$. The SMB redirector will automatically connect to that share if necessary in order to do a remote API call.
2. All remote API operations are done using the Transaction SMB *SMBtrans*.
3. The *smb_name* field of the Transaction SMB is always \PIPE\LANMAN. The server uses this to identify a remote API request. The SMB resembles a normal named pipe operation, which is also done using a Transaction SMB. However, the *smb_setup*[0] field, which would normally contain the desired named pipe operation, is ignored; the \PIPE\LANMAN name field is sufficient to identify a remote API operation.

The arguments for the remote API call are encapsulated in the Transaction request SMB; return values are encapsulated in the Transaction response SMB. In both the request and the response, all binary values are stored in little-endian order, least significant byte first. There are no pad bytes other than those explicitly specified in descriptor strings; therefore, items may be located at an arbitrary byte boundary - there are no alignment restrictions.

The request and response Transaction SMBs contain a parameter section and a data section. The arguments for a remote API call are split into two parts, and placed in these sections of the request Transaction. The Transaction response message contains the results of the call, split between the parameter and data sections of the Transaction response. A number of fields in the Transaction SMB identify the size and location of these sections within the SMB, and also allow a single Transaction request or response to be split into several messages (refer to X/Open CAE Specification, IPC Mechanisms for SMB).

B.3 LMX Access Control Lists Mapping

Access control lists (ACLs) are used by LMX servers running in user-level security mode. Though the implementation of ACLs is outside the scope of the specification the following list is a set of possible access permissions, which is used by LAN Manager implementations.

User-level security allows access permissions to be set for each shared resource (for example, file system subtree, individual file, spooler, device, etc.). Each shared resource has a list of users and groups, with the permissions allowed for each user or group on that resource.

ACL Permissions		
R	read	Permission to read data from a resource and, by default, execute the resource.
W	write	Permission to write data to the resource.
X	execute	Permission to execute the resource.
C	create	Permission to create an instance of the resource (for example, a file); data can be written to the resource when creating it.
D	delete	Permission to delete the resource.
A	change attributes	Permission to modify the resource's attributes (for example, the date and time a file was last modified).
P	change permissions	Permission to modify the permissions (read, write, create, execute and delete) assigned to a resource for a user, group or application.
N	deny access	No permissions.
Y	allow spool requests	

Since the X/Open CAE does not provide an access control list (ACL) mechanism, the usual CAE access control mechanisms should be used instead. Following the principle of least surprise, a mapping is defined for access mechanisms which cannot easily be provided under CAE systems. The CAE access control mechanisms are used to permit interoperability for applications which reside on both PCs and on CAE hosts.

A mapping from (SMB) UID and username/password supplied by the client to CAE User ID (*uid*) and Group ID(s) (*gid*) is established by the *SMBsesssetupX* and will be maintained by the LMX server. The mapped-to CAE User ID and one or more Group IDs are used for all accesses on the CAE system in the usual manner.

The differences between the functionality provided by ACLs and the access control mechanisms for LMX servers described above include:

1. ACL permissions apply to shared resources. This includes file system directories as well as individual files. CAE permissions apply to individual files and directories but are not extended to subtrees.
2. For each resource, ACL permissions can be listed for any number of individual users, for any number of groups, and for anyone else. A CAE file or directory specifies permissions for the owner, one group and everyone else.

The following table shows the mapping between the ACL permissions and CAE permissions:

SMB Permissions		Equivalent CAE Permission	
R	read	r	read
W	write	w	write
X	execute	r	read (Note 1)
C	create	w	write on parent dir
D	delete	w	write on parent dir
A	change attributes		not supportable
P	change permissions		(Note 2)
N	deny access	-	no permissions (Note 3)
Y	allow spool requests		not supportable

Notes:

1. Execute permission for LMX servers requires only read permission, as the client need only be able to read the file before it can execute it.
2. Not an assignable access right. The owner of a file and users with appropriate privileges always have P access and cannot relinquish it; no other user can acquire P access.
3. Not a specific right, but the absence of rights. Note that the privileged user always has all rights and can relinquish none of them.

ACLs could be partially implemented for LMX servers by placing the required checks into the LMX server itself. The list would be used to further restrict (but not grant) access to files and directories beyond the restrictions imposed by the usual CAE access control mechanisms. A client may have access to a resource only if it does not conflict with CAE permissions and if it is specified in the ACL. There may be cases where the ACL indicates that a user should have access, but the CAE security would have to be circumvented to honour it. The access will be denied in accordance with the CAE in these cases. This permits access security to be maintained on both the server and client system equivalently; if a user local on the CAE system is denied access, access should be denied for the user on a client system as well.

X/Open-compliant system implementations which support native ACLs as an enhancement may use that mechanism instead of the normal CAE access control mechanisms if desired, as long as the ACLs do not grant permission where the expected CAE access mechanisms would have denied it.

B.4 Transaction API Request Format

B.41 Parameter Section

The parameter section (*smb_param*) of the Transaction request contains the following:

- API number: 16-bit integer
- parameter descriptor string: null-terminated ASCII string
- data descriptor string: null-terminated ASCII string
- parms: subroutine arguments, as described by the parameter descriptor string
- auxiliary data descriptor string: optional null-terminated ASCII string

The API number identifies which API routine the SMB redirector wishes the LMX server to call on its behalf. A list of API numbers is given in Section B.8 on page 275.

The parameter descriptor string describes the types of the arguments in the data section (*smb_data*), as given in the original call to the routine on the SMB redirector.

The data descriptor string describes the format of a data structure, or data buffer, which is sent to the API routine. The API routine on the SMB redirector is normally given a pointer to this buffer. Note that this descriptor string is also used by the server to determine the format of the data buffer to be sent back from the API call.

The parms field contains the actual subroutine arguments, as described by the parameter descriptor string.

The auxiliary data descriptor string describes the format of a second, auxiliary data structure which is either sent to or received from the API routine, in addition to that defined by the data descriptor string. The data described by this descriptor string is located in the data section (*smb_data*) of *SMBtrans*, immediately following the data described by the primary data descriptor.

B.42 Data Section

The data section (*smb_data*) of the *SMBtrans* request contains the following:

- the primary data buffer, as described by the data descriptor string in the parameter section
- the auxiliary data buffer (optional), as described by the auxiliary data descriptor in the parameter section

B.5 Transaction API Response Format

B.5.1 Parameter Section

The parameter section (*smb_param*) of the *SMBtrans* response contains the following:

- Status: a 16-bit integer. This is the return status as if the requested LAN Manager API routine would be executed on the responder's system. Zero normally indicates success.
- Converter word: 16-bit integer, used by the requestor's system to adjust the pointer in the data section. The use of this field is described below.
- Parms: return parameters, as described by the parameter descriptor string in the request message. Only those parameters which are identified in the parameter descriptor string as being receive pointers (that is, which will be modified by the server) are actually returned here.

B.5.2 Data Section

The data section (*smb_data*) of the *SMBtrans* request contains:

- the primary returned data buffer, as described by the data descriptor in the request message
- the auxiliary data buffer (optional), as described by the auxiliary data descriptor in the request message

B.6 Descriptor Strings

A descriptor string is a null-terminated ASCII string. Descriptor string elements consist of a letter describing the type of the argument, possibly followed by a number (in ASCII representation), specifying the size of the argument. Each item in the descriptor string describes one data element.

B.6.1 Descriptor String Types

The following describes the characters which may be encountered in a descriptor string, and the format of the corresponding data described by the descriptor string.

B Byte

If followed by one or more digits (that is, B13) this refers to an array of bytes. One or more bytes will be located in the corresponding data area. Note that this type will not be found in the parameter descriptor string (that is, it will not be used to describe subroutine arguments), since single bytes cannot be pushed onto the stack by the SMB redirector.

W 16-bit integer

If followed by one or more numbers (that is, W4) this refers to an array of 16-bit integers. One or more 16-bit integers will be located in the corresponding parameter or data area.

D 32-bit integer

If followed by one or more numbers (that is, D3) this refers to an array of 32-bit integers. One or more 32-bit integers will be located in the corresponding parameter or data area.

z Null-terminated ASCII string

The corresponding parameter or data area contains a null-terminated ASCII string. This type has a different meaning when applied to returned data. (See below.)

b Byte pointer

The original argument list or data structure contained a pointer to one (that is, b) or more (that is, b8) bytes at this position. The bytes themselves are located in the corresponding parameter or data area. This type has a different meaning when applied to returned data. (See below.)

w Word pointer

The original argument list or data structure contained a pointer to one (that is, w) or more (that is, w2) 16-bit integers at this position. The integers themselves are located in the corresponding parameter or data area. This type has a different meaning when applied to returned data. (See below.)

d Dword pointer

The original argument list or data structure contained a pointer to one (that is, d) or more (that is, d3) 32-bit integers at this position. The integers themselves are located in the corresponding parameter or data area. This type has a different meaning when applied to returned data. (See below.)

g Receive byte pointer

The original argument list contained a pointer to one (that is, g) or more (that is, g8) bytes at this position, which are to receive return values from the API call. The Transaction request contains nothing at this position in the corresponding parameter or data area; the response message contains data.

h Receive word pointer

Contains data in the parameter section. The original argument list contained a pointer to one (that is, h) or more (that is, h2) 16-bit integers at this position, which are to receive return values from the API call. The Transaction request contains nothing at this position in the corresponding parameter or data area; the response message contains data in the parameter section.

i Receive dword pointer

The original argument list contained a pointer to one (that is, i) or more (that is, i3) 32-bit integers at this position, which are to receive return values from the API call. The Transaction request contains nothing at this position in the corresponding parameter or data area; the response message contains data in the parameter section.

O Null pointer

The original argument list or data structure contained a null pointer at this position. There is nothing stored at this position in the corresponding parms or data area.

s Send data buffer pointer

The original argument list contained a pointer at this position to a data structure containing more data arguments to the API call. This item appears only in a parameter descriptor string. The format of the secondary data structure is described in the data descriptor string (contained in the parameter section of the Transaction request message). The data itself is contained in the data section of the Transaction request message.

T Length of send buffer

The original argument list contained a 16-bit integer argument at this position which specified the length of the send buffer. This item appears only in a parameter descriptor string. No value is placed in the corresponding parameter area.

r Receive data buffer pointer

The original argument list contained a pointer at this position to a data structure which was to be filled in by the API call. This item appears only in a parameter descriptor string. The format of the secondary data structure is described in the data descriptor string (contained in the parameter section of the Transaction request message). The data itself is contained in the data section of the Transaction response message.

L Length of receive buffer

The original argument list contained a 16-bit integer argument at this position which specified the length of the receive buffer. This item appears only in a parameter descriptor string. The corresponding parameter area contains a 16-bit integer specifying the length of the receive buffer.

P Parameter number

The corresponding parameter or data area contains a 16-bit short integer.

e Entries read

The original argument list contained a pointer to a 16-bit integer at this position, which is to receive the number of entries returned by the API call in the receive buffer. The Transaction request contains nothing at this position in the corresponding parameter or data area; the response message contains the numbers of entries returned in the receive data buffer.

N Number of auxiliary structures

This field is only found in data descriptor strings. The presence of the field indicates that there will be auxiliary data sent (if found in a send data descriptor string), or received (if found in a receive data descriptor string). The corresponding data block contains a 16-bit integer specifying the number of auxiliary data structures to be sent (for a send data buffer), or which have been received (for a receive data buffer).

K Unstructured data block

This will normally be the only item in a descriptor string.

F Fill

The corresponding data area contains one (that is, F) or more (that is, F3) fill bytes at this position.

B.6.2 Pointer Types and Returned Data

Lower-case letters are considered pointer types. These pointer types z, b, w and d have a different meaning if they are used to describe returned information. In this case the pointers occur in a data descriptor string or auxiliary data descriptor string and describe data to be returned in the data section (*smb_data*) of the *SMBtrans* response message. In this case the item referred to by the pointer is not the array or string itself, but a 32-bit integer. The high-order 16-bits are to be ignored and the low-order 16-bits contain an offset. The offset subtracted by the converter word points to the array or string within the returned data buffer itself.

The data descriptor describes one instance of the returned data structure. The response buffer may contain several of these data structures, each of which is a fixed size. Together, these make up the fixed-length portion of the returned data area. The returned data buffer may also contain data pointed to by the various pointer types described above. This data may contain strings, and is likely to be of variable length. The fixed-length data is always placed at the beginning of the returned data buffer; the placement of the variable-length data is up to the server.

The responder must place variable-length data at the end of the data buffer and set the pointers accordingly. Since the total length of the data buffer is only known at the end of processing, there may be a gap between the fixed-length data and the variable-length data. To avoid sending this gap across the network the responder may position the variable-length data to a position immediately following the fixed-length data. The pointers in the data descriptor string do not need to get updated if the "converter word" in the response parameter section is set to the value that the requestor must subtract from all pointer values referencing data in the variable-length section.

B.7 Examples

The following examples may help clarify details of the protocol. Some details have been simplified for ease of explanation. Note that the format of some data structures may differ in various versions of LAN Manager.

B.7.1 NetShareDel

This is one of the simplest examples of a remote API call. Suppose an SMB redirector programme does the following call:

```
NetShareDel(SERVER, C, 0);
```

This call deletes the outstanding share C on the server machine SERVER.

The parameter section of the Transaction request message contains:

4: API number for the NetShareDel function.

zW: Parameter descriptor string. Note that the servername argument is not specified in the descriptor. There are two arguments: a string specifying the name of the share to be deleted, and a reserved 16 bit integer MBZ (Must Be Zero).

: Data descriptor string. There is no data buffer in the arguments, so this descriptor string is empty.

parms: The actual subroutine arguments, as described by the parameter descriptor string:

C: A null-terminated string.

0 A 16 bit word.

There is no auxiliary data descriptor string.

The data section of the Transaction request message is empty.

The parameter section of the Transaction response message contains:

return status: (16 bit word.)

converter word: 0 in this case.

return parms: There are no return parameters in this case, so this section will be empty.

The data section of the Transaction response message is empty.

B.7.2 NetShareAdd

This example uses a send buffer:

```
struct share_info_2 buf;
NetShareAdd(SERVER, 2, &buf, sizeof(buf));
```

The parameter section of the Transaction request message contains:

3 API number for the NetShareAdd function.

WsT: Parameter descriptor string.

B13BWzWWWzB9B: Data descriptor string. This corresponds to the elements of the *share_info_2* structure.

parms: The actual subroutine arguments, as described by the parameter descriptor string:

2 The second argument.

Note that there is no data here corresponding to the sT portion of the parameter descriptor string.

There is no auxiliary data descriptor string.

The data section of the Transaction request message contains the contents of the *share_info_2* structure:

- thirteen bytes (from the *shi2_netname* field)
- one byte (from *shi2_pad1*)
- one 16-bit word (from *shi2_type*)
- null-terminated ASCII string, copied from the one pointed to by *shi2_remark*
- one 16-bit word (from *shi2_permissions*)
- two 16-bit words (*shi2_max_uses* and *shi2_current_uses*)
- null-terminated ASCII string, copied from the one pointed to by *shi2_path*
- nine bytes (from *shi2_passwd*)
- one byte (from *shi2_pad2*)

The parameter section of the Transaction response message contains:

- return status (16-bit word)
- converter word: 0 in this case
- return parms: there are no return parameters in this case, so this section will be empty

The data section of the Transaction response message is empty.

B.7.3 NetShareEnum

This example has both return parameters and return data:

```
struct share_info_1 buf[10];
NetShareEnum(SERVER, 1, &buf, sizeof(buf), &nentries, &total);
```

The parameter section of the Transaction request message contains:

0 API number for the NetShareEnum function.

WrLeh: Parameter descriptor string.

B13BWz: Data descriptor string (for returned data, in this case).

parms: The actual subroutine arguments, as described by the parameter descriptor string:

1 Second argument

sizeof(*buf*): This is a send parameter because the server needs to know how much space it has available in which to return results

Note that the other arguments are result parameters, and are thus not passed to the server.

There is no auxiliary data descriptor string.

The data section of the Transaction request message is empty.

The parameter section of the Transaction response message contains:

- return status (16-bit word)
- converter word: (possibly set by server)
- entries returned (16-bit word)
- total number of available entries (16-bit word)

The data section of the response Transaction message contains a number of *share_info_1* structures. The number of such structures is given by the entries returned return parameter. Each structure contains:

- thirteen bytes (the *shi1_netname* field)
- one byte (*shi1_pad1*)
- one 16-bit word (*shi1_type*)
- the *shi1_remark* field. This is a four-byte value. The two low-order bytes contain the offset within the data section of the null-terminated ASCII string. The value may need adjusting: the converter word value must be subtracted from this offset in order to obtain the true offset of the string.
- a possible gap following the fixed-length data. This is up to the server.
- the null-terminated string pointed to by the *shi1_remark* field

B.8 API Numbers

The following are the API numbers used to specify the various remote LAN Manager routines. They are included here so that an implementor can build an LMX server which can handle this class of SMB redirector requests. However, their inclusion in this specification does not imply any X/Open endorsement of these mechanisms as the basis for future X/Open network management functionality. (A routine name beginning with R identifies a routine which gets special handling by the LMX server, rather than simply calling the local version of the routine.)

0 RNetShareEnum
1 RNetShareGetInfo
2 NetShareSetInfo
3 NetShareAdd
4 NetShareDel
5 NetShareCheck
6 NetSessionEnum
7 NetSessionGetInfo
8 NetSessionDel
9 NetConnectionEnum
10 NetFileEnum
11 NetFileGetInfo
12 NetFileClose
13 RNetServerGetInfo
14 NetServerSetInfo
15 NetServerDiskEnum
16 NetServerAdminCommand
17 NetAuditOpen
18 NetAuditClear
19 NetErrorLogOpen
20 NetErrorLogClear
21 NetCharDevEnum
22 NetCharDevGetInfo
23 NetCharDevControl
24 NetCharDevQEnum
25 NetCharDevQGetInfo
26 NetCharDevQSetInfo
27 NetCharDevQPurge
28 RNetCharDevQPurgeSelf
29 NetMessageNameEnum
30 NetMessageNameGetInfo
31 NetMessageNameAdd
32 NetMessageNameDel
33 NetMessageNameFwd
34 NetMessageNameUnFwd
35 NetMessageBufferSend
36 NetMessageFileSend
37 NetMessageLogFileSet
38 NetMessageLogFileGet
39 NetServiceEnum
40 RNetServiceInstall
41 RNetServiceControl
42 RNetAccessEnum
43 RNetAccessGetInfo
44 RNetAccessSetInfo
45 RNetAccessAdd
46 RNetAccessDel
47 NetGroupEnum
48 NetGroupAdd
49 NetGroupDel
50 NetGroupAddUser
51 NetGroupDelUser
52 NetGroupGetUsers
53 NetUserEnum
54 RNetUserAdd
55 NetUserDel
56 NetUserGetInfo
57 RNetUserSetInfo
58 RNetUserPasswordSet
59 NetUserGetGroups
60 NetWkstaLogon
61 NetWkstaLogoff
62 NetWkstaSetUID
63 NetWkstaGetInfo
64 NetWkstaSetInfo
65 NetUseEnum
66 NetUseAdd
67 NetUseDel
68 NetUseGetInfo
69 DosPrintQEnum
70 DosPrintQGetInfo
71 DosPrintQSetInfo
72 DosPrintQAdd
73 DosPrintQDel
74 DosPrintQPause
75 DosPrintQContinue
76 DosPrintJobEnum
77 DosPrintJobGetInfo
78 RDosPrintJobSetInfo
79 DosPrintJobAdd
80 DosPrintJobSchedule
81 RDosPrintJobDel
82 RDosPrintJobPause
83 RDosPrintJobContinue
84 DosPrintDestEnum
85 DosPrintDestGetInfo
86 DosPrintDestControl
87 NetProfileSave

88 NetProfileLoad
89 NetStatisticsGet
90 NetStatisticsClear
91 NetRemoteTOD
92 NetBiosEnum
93 NetBiosGetInfo
94 NetServerEnum
95 I_NetServerEnum
96 NetServiceGetInfo
97 NetSplQmAbort
98 NetSplQmClose
99 NetSplQmEndDoc
100 NetSplQmOpen
101 NetSplQmStartDoc
102 NetSplQmWrite
103 DosPrintQPurge

Appendix C

The X/Open Security Package

The X/Open security package, as defined in this appendix, permits the LMX server to select encryption functions from lists sent by the SMB redirector. This appendix defines some suggested *E*() and *U*() dialect names and the functions associated with those names.

The definitions in this section are not a part of the X/Open specification of the SMB protocols at the present time, and might not become a part of the X/Open specification in the future. Nonetheless, it is recommended that the dialect names defined here are used as defined; if other encryption functions are supported, names defined in this appendix should not be used for them.

C.1 E() Functions

The *E*() function is used to respond to the server and (optional) SMB redirector challenges. It cryptographically combines the challenge string and the password string (in server form, see Section C.2to produce the response string. The function should be chosen so that it is difficult or expensive to derive the password string from the challenge string and response string, even if the cryptographic function is not secret.

The following table gives the *E*() dialect name and a definition for the function to be used if that dialect is selected.

NULL — Value is the password string (in server form), unchanged. Used when the network is known to be secure against eavesdropping (for example, link encryption).

DES[2] — The password string is used as a key to encrypt the challenge string using the DES block mode algorithm. The DES function is applied as described in Appendix D on page 279.

UNIX — The server-form password string is used as input to the well-known UNIX password encryption algorithm[3]. Instead of using a data block of all zeros, the challenge string is used; the salt is two NULL characters.

---

2 U.S. Department of Commerce Data Encryption Standard.

3 Morris, Robert and Thompson, Ken: Password Security: A Case History. Bell Laboratories Technical Memorandum, April 3, 1978. Reprinted in *UNIX Programmers' Manual, Seventh Edition*, Volume 2 page 595. New York: Holt, Rinehart and Winston (1983).

C.2 U() Functions

The *U*() function is used to transform a cleartext password into the form in which it is stored on the server (that is, server-form). Many X/Open-compliant systems store passwords in an encrypted form, and many of these functions are one-way; that is, the transformation from cleartext to cryptotext is not reversible. Negotiation of the *U*() function permits the SMB redirector to reproduce the cryptotext password given the clear password as typed by the user.

Some *U*() functions require additional data aside from the password and username. If the server selects such a *U*() function, it will return the necessary additional data in the *SMBsecpkgX* response. Some LMX server implementations support a mechanism for changing a user's password via some additional protocol; those LMX server implementations should also return any additional data required for that process.

The following table defines *U*() dialect names and the functions to be performed if that dialect is selected. The contents of the *xp_ouinf* and *xp_nuinf* fields of the *SMBsecpkgX* response are also described.

NULL — The server-form of the password is identical to the cleartext form.

UNIX — The well-known UNIX password encryption algorithm is used. The *xp_ouinf* field contains the two-character salt required by the algorithm. If the LMX server supports password changes via protocol, the *xp_nuinf* field should be the new salt to be used if the SMB redirector changes passwords.

Appendix D

SMB Encryption Techniques

D.1 SMB Authentication

The SMB authentication scheme is based upon the server knowing a particular encrypted form of the user's password, the client system constructing that same encrypted form based upon user input, and the client passing that encrypted form in a secure fashion to the server so that it can verify the client's knowledge.

The scheme uses the DES[4] encryption function in block mode; that is, there is a function *E(K,D)* which accepts a 7-byte key *(K)* and 8-byte data block *(D)* and produces an 8-byte encrypted data block as its value. If the data to be encrypted is longer than 8 bytes, the encryption function is applied to each block of 8 bytes in sequence and the results appended together. If the key is longer than 7 bytes, the data is first completely encrypted using the first 7 bytes of the key, then the second 7 bytes, etc., appending the results each time. In other words:

$$E(K_0K_1,D_0D_1)=E(K_0,D_0)\ E(K_0,D_1)E(K_1,D_0)E(K_1,D_1)$$

D.1.1 SMBnegprot Response

The *SMBnegprot* response field *smb_cryptkey* is the result of computing:

C8=E*(P7,S8)*

where:

- *P7* is a 7-byte string which is non-repeating. This is usually a combination of the time (in seconds since January 1, 1970) and a counter which is incremented after each use.
- *S8* is an 8-byte string whose value is `????????` (eight question marks).

D.1.2 SMBtcon, SMBtconX, SMBsesssetupX Requests

The client system may send an encrypted password in any one of these requests. The server must validate that encrypted password by performing the same computations the client did to create it, and ensuring the strings match. The server must compute:

P16=E*(P14,S8)*

and:

P24=E*(P21,C8)*

where:

- *P14* is a 14-byte string containing the user's password in cleartext, padded with spaces.
- *S8* is the 8-byte well-known string (see above).

4 U.S. Department of Commerce Data Encryption Standard.

- *P21* is a 21-byte string obtained by appending 5 null (0) bytes to the string *P16*, just computed.
- *C8* is the value of *smb_cryptkey* sent in the *SMBnegprot* response for this connection.

The final string, *P24*, should be compared to the encrypted string in the request:

- the *smb_passwd* field in *SMBtcon*
- the *smb_spasswd* field in *SMBtconX*
- the *smb_apasswd* field in *SMBsesssetupX*

If they do not match, it is possible the client system was incapable of encryption; if so, the string should be the user's password in cleartext. The server should try to validate the string, treating it as the user's unencrypted password. If this validation fails as well, the password (and the request) should be rejected.

Appendix E

TOP/NetBIOS

This appendix reproduces, in full and unedited, the MAP/TOP Users Group Technical Report Specification of NetBIOS Interface and Name Service Support by Lower Layer OSI Protocols, Version 1.0, September 27, 1989.

MAP/TOP Users Group Technical Report
Specification of NetBIOS Interface and Name Service
Support by Lower Layer OSI Protocols
Version 1.0, September 27, 1989

1 INTRODUCTION

In addition to the universal interoperability TOP products offer, many users have purchased products that conform to proprietary and de facto networking standards. For IBM personal computers and compatibles, a de facto networking standard is the Network Basic Input Output System, or NetBIOS. A majority of popular network applications for these computers require a NetBIOS-compatible interface.

Many vendors recognize this fact and understand the need to preserve investments in these applications while allowing the support of new TOP based applications. Several of these vendors have introduced or plan to introduce TOP products with a NetBIOS-compatible interface.

In order to prevent these vendors from developing separate and incompatible implementations, the TOP NetBIOS Migration Technical Committee has defined a uniform way to support the NetBIOS interface in TOP systems. All products that conform to this specification interoperate with each other, and networks composed of such products support both TOP applications and current PC software packages. The PC applications operate without modification on the local network and, in many cases, as described in section 3.4, across the TOP internetwork. In order to support TOP applications, an implementation must conform to the TOP V3.0 Specification in addition to this NetBIOS support specification.

The specification defined by the TOP NetBIOS Migration Technical Committee consists of this specification. It is logically divided into two parts. The first part defines a mapping of the NetBIOS Interface to ISO Transport Services and Data Link Services. The second part defines a naming protocol for the NetBIOS environment over TOP-recognized subnetworks that support NetBIOS name support services.

Sections 3 through 6 and Appendix I comprise the first part. Section 2, ''Reference Documents,'' specifies the documents that the Technical Committee considers to define the NetBIOS interface and the ISO transport services. Readers should become familiar with these documents, as the remaining sections assume a knowledge of both the NetBIOS interface and ISO transport services and ISO transport profiles.

Section 3 describes the general principles behind the mapping of NetBIOS commands to transport services. Section 4, ''Special Considerations,'' discusses several significant issues in the NetBIOS/transport mapping. Sections 5 and 6 detail the mapping. ''NetBIOS Commands'' describes the mapping of each NetBIOS command to ISO transport services. It identifies the level of support required for each NetBIOS command, and it indicates the specific transport service requests associated with each command. Section 6, ''Transport Service Indications and Confirmations,'' describes the response of the NetBIOS interface to each transport service indication and confirmation. Finally, Appendix I, ''State Tables,'' presents state tables that precisely define the mapping between NetBIOS ''sessions'' and class four transport connections.

Sections 7 through 9 and Appendices II through V define the NetBIOS Name Service Protocol. Appendix VI is provided for future errata or clarifications discovered during product implementation and interoperability testing.

2 REFERENCES AND DEFINITIONS

The first step in defining a mapping between the NetBIOS interface and ISO transport services is to agree on a definition of the NetBIOS interface and OSI services. This section lists the reference documents that the SIG has agreed to use as the definition for NetBIOS and transport.

2.1 The NetBIOS Interface

For the purposes of the mapping specified by this specification, the NetBIOS interface is defined by the first section, ''NetBIOS,'' in the first edition (April 1987) of the IBM publication NetBIOS Application Development Guide (IBM product number 68X2270). When that section directs readers to adapter specific sections for exact details of certain commands (ADAPTER STATUS, for example), those details can be found in this specification. Note that the IBM specification defines the exchange of NCBs (Network Control Blocks - contents and error responses) between a NetBIOS Client and NetBIOS service provider. The contents of the NCBs and error responses are the same for NetBIOS Interfaces for DOS and OS/2 environments; however, the NCB transfer mechanism for these two environments is different and is not covered in this specification.

2.2 OSI Services

- ISO 8072-1986: Open Systems -- Transport Service Definition
- ISO 8072-ADD1: Transport Service Definition -- Addendum 1: Connectionless-Mode Transmission
- ISO 8073-1986: Connection Oriented Transport Protocol Specification
- ISO/DIS 8602: Protocol for Providing the Connectionless-Mode Transport Service
- ISO 8473/N4542: Protocol for Providing the Connectionless-mode Network Service
- ISO 8648: Internal Organization of Network Layer
- ISO 8348, AD1, AD2: Network Service Definition, Connectionless Data Transmission, Network Layer Addressing
- ISO 8802/2: Logical Link Control
- ISO 8802/3: Carrier Sense Multiple Access with Collision Detection (CSMA/CD)
- ISO 8802/4: Token Passing Bus Access Method
- ISO 8802/5: Token Ring Access Method

2.3 Definitions

2.3.1 Reference Model Definitions

This specification makes use of the following concepts defined in the ISO/OSI's Basic Reference Model [ISO 7498]:

DUA ISO Directory User Agent

DSA ISO Directory Service Agent

DIB Directory Information Base

ES End System

IS Intermediate System

LSAP Link Layer Service Access Point

NSAP Network Service Access Point

PDU Protocol Data Unit

psel presentation selector

SNPA Subnetwork Point of Attachment

SNPDU Subnetwork Protocol Data Unit

ssel session selector

TPDU Transport Protocol Data Unit

TSDU Transport Service Data Unit

tsel transport selector

2.3.2 Other Definitions

The following terms/concepts used in this specification, which are not defined in ISO 7498, are as follows:

NCB Network Control Block

NDUA NetBIOS Directory User Agent

NDSE NetBIOS Directory Service Entity

NSP NetBIOS Name Service Protocol

NSPDU NetBIOS Name Service Protocol Data Unit

2.3.3 Service Conventions Definitions

This Protocol Specification makes use of the following terms from the OSI Service Conventions Technical Report (ISO TR 8509):

1. Service provider
2. Service user

2.3.4 Additional Definitions

For the purposes of this specification, the following definitions apply:

1. Group Name: a name which can be shared among multiple owners; a name which is not unique. This definition derives from the NetBIOS group name concept, rather than from the ISO/CCITT group entry.
2. Local Matter: a decision made by a system concerning its behavior in the Directory System that is not prescribed or constrained by this specification.
3. Protocol Address: the complete protocol address of an object or entity, consisting of its transport address.
4. Byte and Octet: used interchangeably in the specification.

3 GENERAL PRINCIPLES

Before embarking on a detailed description of the mapping between the NetBIOS interface and ISO transport services, it is important to understand several general principles upon which this specification is based. The NetBIOS interface is best supported at the ISO transport layer; NetBIOS ''sessions'' best map to class 4 transport connections, and NetBIOS Datagrams best map to connectionless transport data requests except in the case of broadcast datagrams (broadcast name services) where a Data Link level mapping is required. The NetBIOS general commands, with one exception, do not require

any exchange of peer-to-peer protocol data units. The following subsections discuss each of these principles in more detail.

3.1 NetBIOS Supported on a Transport Service

The best level in the OSI reference model at which to map the NetBIOS interface is the level whose services most closely parallel the services offered by the NetBIOS interface. That is the OSI transport level. The NetBIOS interface requires reliable, sequenced data delivery, a service only available at the transport level and above. The NetBIOS interface, however, does not provide upper level services such as token management, synchronization and activity management. The only OSI level above the network level and below the session level is, of course, the OSI transport level, and it is to this level that the NetBIOS interface best maps.

Readers should be cautioned that the NetBIOS interface definition (see above) often refers to the NetBIOS interface as a ''session'' level interface. These references exist because the protocols that support the original NetBIOS interface (on the original PC Network Adapter) were developed before the OSI reference model was widely understood. The highest level protocols on the adapter were called ''session'' protocols despite the fact that they do not provide OSI session services. Throughout this specification, terms which refer to the NetBIOS view of a ''session'' will be placed in quotation marks. Terms which refer to the OSI view of a session will remain unquoted.

In addition to its data transfer services, NetBIOS provides name service support. The specific naming services NetBIOS provides differ fundamentally from the current ISO directory services. No reasonable mapping between NetBIOS name support and ISO directory services exists, so NetBIOS name support does not affect the choice of protocol level at which to map the NetBIOS interface. A protocol that provides NetBIOS naming services is specified in the Sections 7 through 9.

Choosing to map NetBIOS to the transport level does provoke another concern: the NetBIOS assumption of confirmed data delivery. NetBIOS data transfer between ''sessions'' is a confirmed service, while ISO transport services provide only unconfirmed data delivery (see ''Confirmed Data Delivery'' in the following section).

One important consequence of mapping the NetBIOS interface to transport services is that NetBIOS ''addresses'' equate to transport selectors. A NetBIOS ''address'' is a NetBIOS name; NetBIOS names correspond to transport selectors. The transport address is the combination of a network service access point (NSAP) address and a transport service access point selector (T-Selector). The NSAP address for a name is an NSAP address on the network node at which the name exists; the T-Selector for a name is equal to the full NetBIOS name itself. Since the NetBIOS interface requires that names be exactly sixteen characters long, T-Selectors used by NetBIOS names are also sixteen bytes long. The correspondence between a NetBIOS name and a transport address (an NSAP address and T-Selector pair) is detailed in part two of this specification.[5]

3.2 NetBIOS ''Sessions'' as Transport Class Four Connections

Since the NetBIOS interface best maps to the transport level, NetBIOS ''sessions'' correspond to transport connections. Furthermore, since NetBIOS ''sessions'' require reliable data delivery with automatic error detection

5 Sections 7-9 and Appendices II-V

and recovery, when operating over a connectionless network service, they require class four (TP4) transport connections. Since this specification assumes a connectionless network service, the NetBIOS ''session'' support commands map to TP4 services. LISTEN and CALL commands establish a TP4 connection; SEND, CHAIN SEND, RECEIVE and RECEIVE ANY commands transfer data on that connection, and HANG UP commands terminate the connection. The ''NetBIOS Commands'' and ''Transport Service Indications and Confirmations'' sections of this specification describe the operations required to support each of these commands. Appendix I, ''State Tables,'' details the mapping between ''sessions'' and TP4 connections.

3.3 NetBIOS Datagrams as Connectionless Transport Unitdata Requests

Data transfer with NetBIOS datagrams, unlike NetBIOS ''sessions'', is a connectionless mode of transmission. Naturally, therefore, NetBIOS datagrams correspond to data transfers using the connectionless mode transport service. NetBIOS datagrams may be sent as broadcast datagrams or as multicast datagrams to group names. In order to support broadcast datagrams and datagrams to group names, the NetBIOS interface requires some form of multicast or broadcast addressing. Currently, the ISO transport and network layers do not support multicast or broadcast network addresses.

TOP support for multicast and broadcast addressing is only available through the ISO 8802 link level protocols, so broadcast datagrams and datagrams to group names must use link level addressing. Section 4.3 of this paper, ''Broadcast Datagrams and Datagrams to Group Names,'' details the addressing techniques used.

Because NetBIOS datagrams may contain as many as 512 bytes, the NetBIOS interface requires the lower level services to support a datagram size able to include both the 512 bytes of data and header information for NetBIOS, Transport, Network and Data link Layers. This requires a minimum frame size of 650 octets.

Detailed documentation of the support required for SEND DATAGRAM, SEND BROADCAST DATAGRAM, RECEIVE DATAGRAM and RECEIVE BROADCAST DATAGRAM can be found in the ''NetBIOS Commands'' and ''Transport Service Indications and Confirmations'' sections below.

3.4 Guidelines and Constraints

1. There are three levels of NetBIOS interface services which imply different constraints on the networked NetBIOS based application interconnectivity, see Figure 2.

   - Level A - NetBIOS Connection Services: These services rely on the Connection Oriented Transport and Connectionless Network Protocols, thus following full communication beyond the local network.
   - Level B - NetBIOS Connection and Point-to-Point Datagram Services: These services are a superset of Level A services. As they rely on the Connectionless Network Protocol, communication is possible beyond the local subnetwork. However as the Connectionless Transport is used, the loss of NetBIOS Datagram, if it occurs, would not be recovered from by the Transport Layer.
   - Level C - Extended NetBIOS Services: These services are a superset of Level B services which adds the support of the NetBIOS broadcast and multicast datagram services. As these added services do not use the Connectionless Network Protocol, no direct communication (i.e., no OSI Routing) is possible beyond the local subnetwork. As a consequence any NetBIOS based application requiring Level C Services will have to be distributed only within a single Subnetwork.

2. The use of NetBIOS Name Services and the manner in which they are distributed imply the following constraints.

   a. Name Services scope support based on multicast mechanism is limited to a local subnetwork (the same as NetBIOS native networking).

   b. The expected way to extend the local scope of NetBIOS naming is to integrate the NetBIOS Name Servers into an OSI Directory Services Environment.

3.5 NetBIOS General Commands

Normally, the NetBIOS general commands do not require any peer-to-peer protocol support. For example, no mapping to an ISO protocol is required for RESET, CANCEL, UNLINK and SESSION STATUS commands. The type of support required for each of these commands is detailed below in ''NetBIOS Commands.''

However, one general command, ADAPTER STATUS, sometimes requires communication with a remote system. When the ADAPTER STATUS specifies a remote name, the local system must communicate with the remote system in order to obtain the status. This communication uses the naming protocol defined in NetBIOS Name Service Protocol Specification, so complete documentation of this procedure can be found Appendix V.

The ADAPTER STATUS command also returns a buffer with fields that only apply to specific adapters. The values that adapters conforming to this specification should use for these fields are stipulated in ''ADAPTER STATUS'' in Appendix V.

4 SPECIAL CONSIDERATIONS

A straightforward mapping from the NetBIOS interface to ISO transport services does not resolve all the major NetBIOS/transport issues. It does not specify how transport services provide zero octet sends, confirmed data delivery, how they prevent data loss during hang ups, how they deliver broadcast datagrams and datagrams to group names, how they affect NetBIOS timeouts, how they resolve connections between group names, or how they support permanent node names. This section discusses each of these topics.

This NetBIOS ''Session'' (mapping) Protocol resides above the transport layer and makes use of the services provided by the transport protocol. This protocol specifies use of two-octet NetBIOS headers for data transfer requests (TSDUs). The headers are fixed and always present.[6] The specific values for the header are given in Table 1. The headers are used to solve the issues of zero octet length messages and data loss during hang ups, as described in the following subsections. The most significant octet is transmitted first.

Value	Description
0100H	normal data (connection-oriented or connectionless)
0200H	close request (connection-oriented only)
0300H	close response (connection-oriented only)

TABLE 1. NetBIOS Header Values

6. Note that NetBIOS ''Session'' header is applied to the first TPDU only, and not all the TPDUs when a TSDU is segmented into multiple TPDUs.

Note: A TSDU with an invalid header will be ignored.

Once the transport circuit is established, all the connection oriented data TSDUs generated by the NetBIOS interface/protocol layer will contain a two octet fixed header, carrying NetBIOS opcode as defined above. Additionally all non name service NetBIOS datagram TSDUs contain the two octet fixed header with value 0100H. Note, however, that this does not apply to TSDUs generated by the Name Service Protocol described in sections 7 through 9.

Also, note that the header applies to TSDUs, not TPDUs or TIDUs.

4.1 Zero Length Data and Normal Data Transfer

The NetBIOS data transfer requests are mapped into data TSDUs with NetBIOS header of 0100H for normal data as well as zero length data. Implementations must evaluate the length of TSDUs to determine whether or not it has zero length ''user data''.

4.2 Confirmed Data Delivery

The issue with mapping the NetBIOS interface to transport services is guaranteeing data delivery on ''sessions''. When a NetBIOS SEND or CHAIN SEND command completes, the local user is assured that the remote user has actually received the data. The ISO transport services, however, provide no indication to the sender of actual data delivery; they do not have a T-DATA confirmation primitive. Software implementing a NetBIOS interface does not necessarily know when to indicate that a SEND command has completed.

This behavior can create a problem because, in some application programs, the sender may take actions based on an assumption that the receiver has possession of the data. Taking these actions before the receiver actually does have the data may cause the application program to fail. Fortunately, most NetBIOS application programs do not require true confirmed data delivery; they only need assurance that data is not lost when the ''session'' is closed. This specification, therefore, provides a means of preventing data loss during hang up (see below). Implementations are, of course, free to add a confirmed data delivery service during normal data transfer. The details of such a service are a local matter.

4.3 Data Loss During Hang Up

Because the NetBIOS interface cannot depend on ISO transport services to guarantee data delivery at all times, the interface must prevent data loss during hang up. The NetBIOS definition states that a HANG UP command does not complete until all outstanding SEND and CHAIN SEND commands on the ''session'' have completed (either successfully or unsuccessfully). Because NetBIOS confirms data delivery by completing the SEND command, NetBIOS users are guaranteed that either all data will be delivered prior to the hang up, or that an unsuccessful SEND or CHAIN SEND completion will alert them to data that could not be delivered.

The transport T-DISCONNECT request, on the other hand, is not graceful. It does not wait for all data sent to be delivered to the user. Without confirmed data delivery, the transport user has no way of knowing whether or not data has been delivered to the receiver before the disconnect completes.

To prevent data loss, the NetBIOS interface must delay the transport disconnect until all data has been delivered to the user. To find out when all data has been successfully delivered, the interface that wishes to hang up sends a simple close request packet to the remote interface. This close request is sent ''in stream'' as a normal data TPDU with NetBIOS opcode of 0200H. When the remote interface has received all of these data messages followed by a ''close request'' message and successfully delivered data

messages to the remote user, it sends a ''close response'' back to the local interface, with NetBIOS opcode of 0300H. When the local interface receives the close response, it knows that all data has been delivered. At that point it issues a T- DISCONNECT request and completes the HANG UP command.

The close request and close response are each sent as a single data TSDU with two octet of transport data for the NetBIOS header. The appropriate headers are given in Table 1.

The case of close request collision is handled in a fashion similar to OSI Session Protocol. Under these circumstances, close indication is given to each end point. The action taken by each end point depends on its role at the time the connection was established. The end point which originally issued the connect request should immediately send a close response. The end point which originally accepted the connect request should not send its close response until a close response has been received from the other end point.

In addition to sending the close request, the NetBIOS interface initiating a hang up starts a timer. If that timer expires before the interface receives a close response, the ''session'' is terminated abnormally and the interface immediately issues a T-DISCONNECT request. The interface also aborts the ''session'' if it receives a T-DISCONNECT indication without having sent a close response.

The close operation is detailed in the state tables of Appendix I.

4.4 Broadcast Datagrams and Datagrams to Group Names

An important issue in mapping the NetBIOS interface directly to transport services is NetBIOS datagrams to group names and NetBIOS broadcast datagrams. In order to support broadcast datagrams and datagrams to group names, the NetBIOS interface requires some form of multicast or broadcast addressing. Currently, however, the ISO transport and network layers do not support multicast or broadcast network addresses. These datagrams, therefore, cannot be transferred by the current ISO transport or network level protocols. Note that here ''broadcast'' refers to NetBIOS BROADCAST DATAGRAM commands, not true media level broadcasts.

ISO support for multicast and broadcast addressing is available through the ISO 8802 link level protocols, so broadcast datagrams and datagrams to group names may be transferred by the link level. When the NetBIOS interface wishes to send either type of multicast datagram, it directs the datagram to TOP/NetBIOS Specific Media Access Control (MAC) Multicast Address [see Appendix IV][7]. The interface uses the node's normal MAC address as the source MAC address. Address recommendations for Token Ring networks are provided in Appendix IV ''Well Known Addresses''.

In order to differentiate these NetBIOS datagrams from non- NetBIOS ''pure'' OSI traffic, the interface also uses a special Logical Link Control (LLC) service access point for NetBIOS multicast datagrams. By using a separate LSAP, nodes avoid the possibility of conflict between invented NetBIOS protocol for multicast/broadcast datagrams and an ISO multicast/broadcast service which might be provided through the regular ISO LSAP in the future. The specific LLC service access point defaults to the recommended value of ECH[8]; however, conforming implementations must give users the ability to

7. The Specific Multicast Address for IEEE 802.3 is 09.00.6A.00.01.00. This MAC address is part of the block of Ethernet addresses assigned to AT&T; AT&T has agreed to contribute it to the NetBIOS Special Interest Group This address must be configurable

8. This value of LSAP is from public domain, and this value must be configurable.

configure it to any other value. The selected service access point serves as both the source and destination LLC address. Note that all the nodes on the subnetwork have to be configured with the same LSAP value for this purpose; inconsistent LSAP values will prevent intercommunication.

This addressing allows NetBIOS to send and receive multicast datagrams, but the interface requires additional addressing information. NetBIOS must know the source and destination names for each datagram sent to a group name, and it must know the source name for each broadcast datagram. For point-to-point communications, this information is normally available through the T-Selector.

In order to provide complete addressing information, NetBIOS multicast datagrams continue to use the connectionless transport and connectionless network protocols. Thus each datagram still has local and remote T-Selectors associated with it, and, as is the case with normal datagrams, these T-Selectors indicate the local source and destination names. At the network level, multicast datagrams use the same source NSAP as normal datagrams; the destination NSAP, however, is a special NSAP which indicates the destination is a multicast NSAP. The recommended NSAP address is 49.nn.nn.09.00.6A.00.01.00.01, where [nn.nn=00.00] represents the subnetwork number. Note that these datagrams use a special LLC service access point and this NSAP address is not reported in the ES-IS protocol. Thus, strict TOP-conformant (i.e., non-NetBIOS) implementations of the ISO Connectionless Network Protocol which do not support this special multicast NSAP need not send or receive these datagrams. See Appendix VI for all the ''well known addresses.''

Strictly speaking, NetBIOS multicast datagrams have their own protocol stack invented by the NetBIOS SIG for operation over the ISO datalink layer. This stack, which includes the connectionless transport layer and full network layer (not the inactive subset) protocols, separates from the standard stack at the LLC level, and the two stacks are kept separate by distinct LLC service access points. Implementations, of course, are free to combine these two logical stacks into a single physical stack. Such a combination allows efficient use of common code. A protocol model of this NetBIOS implementation under OSI environment is given in Figure 1[9]. Figure 2 provides a NetBIOS architecture based on the protocol model presented in Figure 1.

As an important consequence of using link level addressing, NetBIOS sacrifices the ability to send multicast datagrams across the TOP internet. NetBIOS broadcast datagrams and datagrams to group names are restricted to the local subnetwork.

Another issue with NetBIOS broadcast datagrams (but not datagrams to group names) is the selection of a remote T- Selector to which they should be sent. Since there is no destination name for these datagrams, the remote T-Selector cannot be determined from the name as it is for normal datagrams. Broadcast datagrams, therefore, use a destination T-Selector equal to the ASCII value for an asterisk (2AH) followed by fifteen bytes equal to the ASCII value for a space (20H).

Table 2 summarizes the addresses NetBIOS requires for multicast and point-to-point datagrams. The actual recommended value for the TOP/NetBIOS Multicast and Functional address are defined in Appendix IV.

9. The dotted line in Figure 1 indicates the boundary between OSI Standard Protocol and NetBIOS specific support protocol.

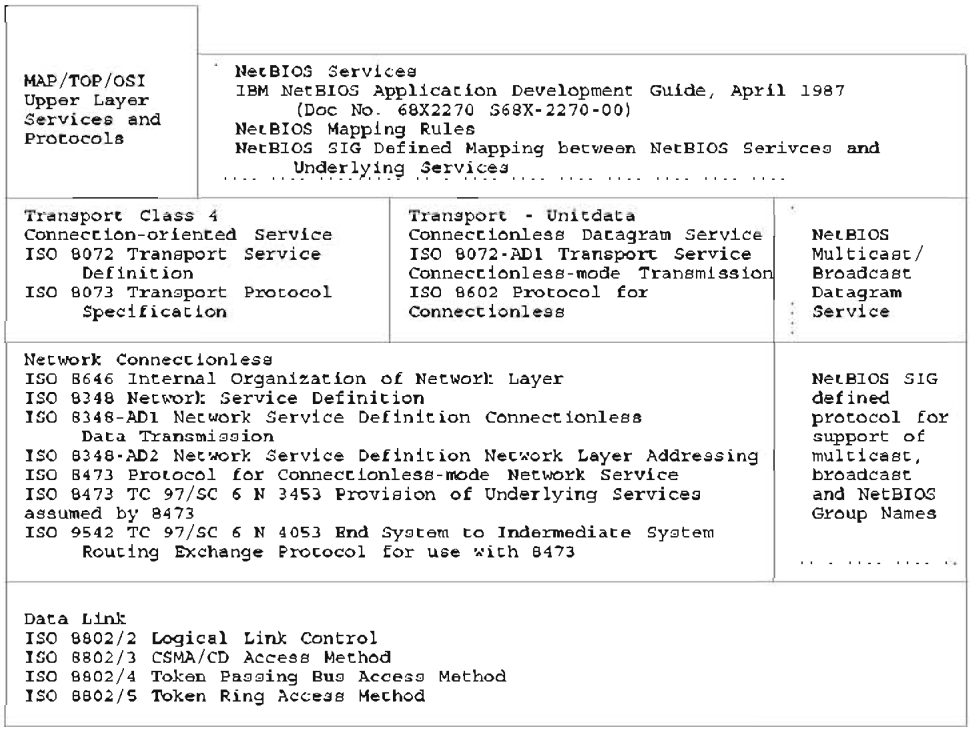


Figure 1. NetBIOS Protocol Model

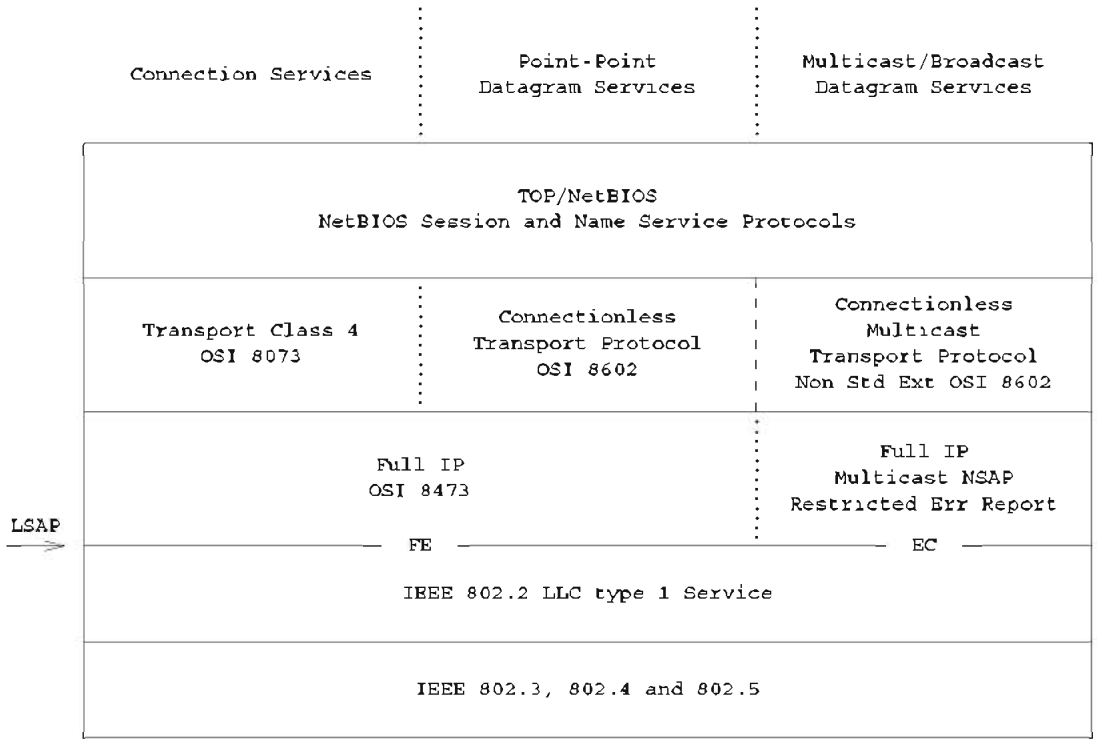


Figure 2. OSI/NetBIOS Architecture

Type	Point-to-Point	Multicast
Source MAC address	source adapter's	source adapter's
Dest. MAC address (CSMA/CD)	dest. adapter's	TOP/NetBIOS Multicast Address
Dest. MAC address (Token Ring)	dest. adapter's	TOP/NetBIOS Functional Address
Source LLC SAP	FEH	ECH
Destination LLC SAP	FEH	ECH
Source NSAP	source adapter's	source adapter's
Destination NSAP	dest. adapter's	multicast NSAP
Source T-Selector	source name	calling name
Dest. T-Selector	destination name	called name or "*<15 sp>"

TABLE 2. Default NetBIOS Addresses

4.4.1 Network Header - Multicast NPDUs

The network header for the PDUs for multicast traffic will be as per OSI 8473 Specification with the error bit turned off.

4.5 Send and Receive Timeouts

The NetBIOS interface defines send and receive timeouts for its ''sessions''. These timeouts limit the amount of time the interface should wait for a SEND, CHAIN SEND or RECEIVE command to complete. Application programs that use these timeouts usually base their values on local subnetwork ''sessions''. Since the original NetBIOS does not support internetworking, application programs are unlikely to account for internetwork transit delay when they specify a send or receive timeout value. Implementations that map the NetBIOS interface to ISO transport services should adjust the send and receive timeout values appropriately for ''sessions'' in case they cross subnetwork boundaries. The definition of ''appropriately'' in this case is left as a local matter.

4.6 ''Sessions'' with Group Names

Another consideration in the mapping of NetBIOS to transport is the establishment of ''sessions'' with group names. This specification requires support of ''sessions'' between group names. NetBIOS LISTEN and CALL commands with group names for the local name are accepted by the interface. The LISTEN command responds to any T-CONNECT indication specifying the correct T-Selector, and the CALL command results in a T-CONNECT request with the appropriate local T-Selector. Additionally, the interface accepts LISTEN and CALL commands with group names for the remote name. The LISTEN command matches any T-CONNECT indication with the appropriate remote T-Selector, and the CALL command results in a T-CONNECT request with a remote T-Selector equal to the remote group name. In all cases, communication occurs through standard ISO protocols attached to the normal ISO LSAP.

The only significant concern in connecting group names is the NSAP address used in a T-CONNECT request when an application program calls a remote group name. That NSAP address should be the specific address (i.e., not generic or group address) of one system on which the group name exists. When the group name exists on more than one system, the choice of which remote NSAP address to use is, for the purposes of this specification, arbitrary. In cases where an NDSE receives multiple responses, it is a local matter how one is chosen for use. In the case where an NDUA is responding to an NDSE, the NDUA may choose one address to put into the response PDU. The approach to be used to make the choice is a local matter.

4.7 Permanent Node Names

A permanent node name, which consists of ten octets of zeros followed by six octets of Mac address, should be treated the same as any other NetBIOS name. Calls to permanent node names, for example, should attempt to discover the address of the remote name just as they would for normal names. The six non-zero bytes in a permanent node name cannot be assumed to correspond to the Ethernet or MAC-layer address of the adapter (but may actually be). Those same six bytes, however, should be returned as the unit identification number by the ADAPTER STATUS command (see below).

An adapter must, of course, successfully register its permanent node name with the NetBIOS naming services each time it is initialized.

5 NetBIOS COMMANDS

The previous three sections specify a definition for the NetBIOS interface and ISO transport services, outline the general principles for mapping the two to each other, and discuss significant complications arising from the mapping. This section begins a detailed description of that mapping. It identifies the level of support required for each NetBIOS command, and it indicates the specific transport service requests and responses associated with each command. NetBIOS commands not listed in this section (TRACE and FIND NAME, for example) are not part of the NetBIOS interface as defined in section 2.1. This specification does not specify support for these additional commands.

Most NetBIOS commands require some initial validation before the interface accepts them. This initial validation may include verifying that the correct adapter was specified, that a name has a valid format, that a local name exists, that a name number is valid, that a ''session'' exists, etc.. The NetBIOS interface definition described in section two, of the referred IBM document, includes an adequate description of this validation. Consequently, this specification omits any description of the validation procedures. Conforming implementations, however, must perform validation for each command as it is described in the NetBIOS interface definition.

Conforming implementations must be able to process NO WAIT commands issued from a post routine call by NetBIOS when a previous NO WAIT command has completed.

5.1 RESET

Implementations conforming to this standard accept and process RESET commands. A RESET command resets the adapter status, deletes all names except the permanent node name, and terminates all ''sessions''. It does not reset traffic and error statistics.

The only protocol interactions resulting from a RESET command are requests to delete NetBIOS names and T- DISCONNECT requests to close NetBIOS connections. Implementations need not delete names belonging to non- NetBIOS programs or protocols, nor must they close non- NetBIOS connections. This specification does not attempt to specify the operation of non-NetBIOS names and connections.

The RESET command may also specify the number of commands and the number of ''sessions'' to be supported by the adapter. Conforming implementations must accept and process these parameters. If the RESET command specifies a value of zero for either parameter, the minimum number of sessions and the number of commands are configured to implementation specific values.

5.2 CANCEL

Conforming implementations accept and process CANCEL commands. Processing is identical to that specified in the NetBIOS definition. Cancelling a CALL, SEND, CHAIN SEND or HANG UP commands results in an immediate T-DISCONNECT request on the affected connection. Cancelling any other valid command does not require any protocol interaction.

5.3 ADAPTER STATUS

ADAPTER STATUS commands for both local and remote adapters are accepted and processed. Local status requests need not require protocol interaction (details are left up to individual implementations); remote status requests, however, use the services of the NetBIOS naming protocol. The format of adapter status request/response is given in Appendix III.

When responding to an ADAPTER STATUS command, the NetBIOS interface fills in a buffer with appropriate status information. Several fields within that buffer apply only to specific adapters or specific network topologies. Since it is not the intent of this specification to restrict implementations to these few specific technologies, this specification must leave the exact support of the ADAPTER STATUS command as a local matter. Implementations should strive to use values for the status fields as close as possible to the values indicated below.

- Unit identification number: The six non-zero bytes of the adapter's permanent node name. These bytes do not necessarily form the Ethernet or MAC layer address of the adapter.
- External option status: One byte whose value is a local implementation choice.
- Results of last self test: One byte indicating the results of the last self-test. A binary value of 128 (80H) indicates that the test was successfully passed.
- Software version: Two bytes containing binary values for the major and minor version number of this specification to which the adapter conforms. The version number for this specification is 1.0.
- Duration of reporting period: Two bytes whose value is a local implementation choice. It is suggested that if the interface reports the MAC statistics indicated by the next eight items, this field contains the binary value of the time, in minutes, since the adapter began recording the statistics. This value rolls over after reaching a value of <2**16-1> minutes. If the interface does not report MAC statistics, it is suggested that this field contains zero.
- Number of CRC errors received: Two bytes whose value is a local implementation choice. It is suggested that they either contain zero or the binary value of the number of MAC-layer packets (frames) with CRC errors received by the adapter. This value is not necessarily restricted to NetBIOS frames, and it does not roll over after reaching <2**16-1> errors.
- Number of alignment errors received: Two bytes whose value is a local implementation choice. It is suggested that they either contain zero or the binary value of the number of MAC- layer packets (frames) with alignment errors received by the adapter. This value is not necessarily restricted to NetBIOS frames, and it does not roll over after reaching <2**16-1> errors.
- Number of collisions encountered: Two bytes whose value is a local implementation choice. It is suggested that they either contain zero or

the binary value of the number of MAC-layer collisions detected by the adapter. This value is not necessarily restricted to NetBIOS frames, and it rolls over after reaching <2**16-1> collisions.

- Number of unsuccessful transmissions: Two bytes whose value is a local implementation choice. It is suggested that they either contain zero or the binary value of the number of MAC- layer packets (frames) whose transmission was aborted by the adapter. This value is not necessarily restricted to NetBIOS frames, and it rolls over after reaching <2**16-1>.
- Number of successfully transmitted packets (frames): Four bytes whose value is a local implementation choice. It is suggested that they either contain zero or the binary value of the number of MAC-layer packets (frames) successfully transmitted by the adapter. This value is not necessarily restricted to NetBIOS frames, and it rolls over after reaching <2**32-1> packets.
- Number of successfully received packets: Four bytes whose value is a local implementation choice. It is suggested that they either contain zero or the binary value of the number of MAC-layer packets (frames) successfully received by the adapter. This value is not necessarily restricted to NetBIOS frames, and it rolls over after reaching <2**32-1> packets.
- Number of retransmissions: Two bytes whose value is a local implementation choice. It is suggested that they either contain zero or the binary value of the number of MAC-layer packets (frames) retransmitted by the adapter. This value is not necessarily restricted to NetBIOS frames, and it rolls over after reaching <2**16-1> retransmissions.
- Number of times the receiver exhausted its resources: Two bytes whose value is a local implementation choice. It is suggested that they either contain zero or the binary value of the number of times the receiver did not have sufficient buffers to receive an incoming MAC-layer packet. This value is not necessarily restricted to NetBIOS frames, and it does not roll over after reaching <2**16-1>.
- Reserved for internal use: Eight bytes whose value is a local implementation choice.
- Free NCBs: Two bytes containing the binary value of the number of additional NetBIOS commands the adapter can currently accept.
- Configured maximum NCBs: Two bytes containing the binary value of the maximum number of commands that the adapter can support, as configured by the last RESET command or initialization.
- Maximum number of NCBs: Two bytes containing the binary value of the maximum number that the adapter can accept in the next RESET command for the ''maximum number of commands supported'' parameter.
- Reserved for internal use: Four bytes whose value is a local implementation choice.
- Pending sessions: Two bytes containing the binary value of the number of currently active or pending ''sessions''.
- Configured maximum sessions: Two bytes containing the binary value of the maximum number of ''sessions'' that the adapter can support, as configured by the last RESET command or initialization.
- Maximum number of sessions: Two bytes containing the binary value of the maximum number that the adapter can accept in the next RESET command for the ''maximum number of sessions supported'' parameter.
- Maximum ''session'' data packet size: Two bytes containing the binary value, in octets, of the maximum TPDU size supported by the adapter, minus

the maximum TP header size.

- Quantity of names in local name table: Two bytes containing the binary value of the current number of NetBIOS names claimed by the adapter. This value does not include the adapter's permanent node name, nor does it include any names used by programs or protocols other than the NetBIOS interface. This number also indicates the maximum number of name entry pairs (the next two fields) which can follow.
- Name: the sixteen byte NetBIOS name.
- Name status: Two bytes, the first of which contains the binary value for the NetBIOS name number, and the second of which contains the name's status. The most significant bit of this second byte indicates whether the name is a unique name (if the bit is clear) or a group name (if the bit is set). The three least significant bits of the status denote the condition of the name. The remaining bits of the name status are undefined, and their values are a local implementation choice. The following list summarizes the values for this field.

  0xxxxxxx name is a unique name

  1xxxxxxx name is a group name

  xxxxx000 name is trying to register

  xxxxx100 name is registered

  xxxxx101 name is de-registered

  xxxxx110 name has been detected as a duplicate

  xxxxx111 name has been detected as a duplicate and is pending de-registration

5.4 UNLINK

This specification does not provide support for the UNLINK command (nor, in fact, for remote program load). A conforming implementation's response to an UNLINK command is left as a local choice.

5.5 ADD NAME

Conforming implementations accept and process ADD NAME commands. The NetBIOS interface translates the ADD NAME command into an appropriate request for the NetBIOS naming services. When the interface receives a confirmation from the naming services, it translates the confirmation's result to an appropriate NetBIOS return code and completes the ADD NAME command. Details of name registration can be found in NetBIOS Name Service Protocol (Section 9).

5.6 ADD GROUP NAME

Conforming implementations accept and process ADD GROUP NAME commands. The NetBIOS interface translates the ADD GROUP NAME command into an appropriate request for the NetBIOS naming services. When the interface receives a confirmation from the naming services, it translates the confirmation's result to an appropriate NetBIOS return code and completes the ADD GROUP NAME command. Details of name registration can be found in NetBIOS Name Service Protocol (Section 9).

5.7 DELETE NAME

Conforming implementations accept and process DELETE NAME commands according to the NetBIOS interface definition. If the name has active ''sessions'', the interface marks the name for eventual deletion and returns the DELETE NAME command with a return code of ''command completed, name has active

''sessions'' and is now de-registered'' (0FH). When all the active ''sessions'' have closed or aborted, the interface actually deletes the name. If the name has pending commands other than active ''session'' commands, those commands are returned immediately with a ''name was deleted'' (17H) completion.

When the NetBIOS interface deletes the name (either immediately or after all active ''sessions'' have closed), it sends an appropriate request to the NetBIOS naming services. Details of name deletion can be found in NetBIOS Name Service Protocol (Section 9).

5.8 CALL

Conforming implementations accept and process CALL commands. When it receives a CALL command, the implementation first finds the transport address corresponding to the remote NetBIOS name. To find this address, it sends a resolve name request to the naming services. If the naming services cannot discover the name's address, the interface completes the CALL command with a return code of ''no answer (cannot find name called)'' (14H).

If the name resolution is successful, the interface continues processing by attempting to establish a transport connection with the remote system. It formulates an appropriate T-CONNECT request to pass to the transport services. The called transport address for the indication consists of the NSAP address of the node on which the remote name resides, along with a T-Selector equal to the remote name. If the remote name is a group name, the NSAP address is that of one node on which the remote name resides; it is not the NetBIOS multicast NSAP address. If the remote group name exists on more than one node, the choice of which NSAP address to use is arbitrary (see ''Sessions with Group Names'' in section 5.6 above).

When the interface receives a T-CONNECT confirmation, it completes the CALL command successfully. If the interface receives a T-DISCONNECT indication instead, it examines the reason code of the indication. If the remote TS-user initiated the disconnect, the interface completes the call with a ''session open rejected'' (12H) return code. If the transport provider initiated the disconnect, or name resolution fails, the interface completes the call with a ''no answer (cannot find name called)'' (14H) return code.

5.9 LISTEN

Conforming implementations accept and process LISTEN commands. When the implementation receives a LISTEN for a valid local name, it holds onto the command until it receives an appropriate T-CONNECT indication (see following section). At that point, the interface completes the LISTEN command. The interface may also complete the LISTEN command if it is cancelled or if the local name is deleted; in these cases the LISTEN completes unsuccessfully.

5.10 HANG UP

Conforming implementations accept and process HANG UP commands. When an implementation receives a HANG UP command, it immediately terminates any pending RECEIVE commands and one RECEIVE ANY command for the ''session'' with a ''session closed'' (0AH) return code. Any subsequent RECEIVE, SEND, CHAIN SEND, or even HANG UP commands for the ''session'' are also immediately terminated with this same return code. The local interface also starts a timer as soon as it receives a HANG UP. If the HANG UP has not completed when this timer expires, the interface aborts the ''session''.

It sends a close request to the remote interface and waits for a close response. When the interface receives the close response, it successfully completes the HANG UP command and issues a T-DISCONNECT request.

If the interface receives a close request after it has sent one, then a ''close collision'' has encountered. Under such situation, if the local interface is the initiator of the ''session'', it will send a close response and then wait for a close response, and the normal HANG UP process continues as described above.

However, if the local interface is the acceptor of the ''session'', in a ''close collision'' situation, it will not issue a close response until it has received one. Following that it will wait for a T-DISCONNECT indication in order to complete the HANG UP process successfully.

If the interface receives a close request or a T-DISCONNECT indication before the close response, it aborts the ''session'' by completing all pending commands with ''session ended abnormally'' (18H) return codes, and, if necessary, issuing a T-DISCONNECT request.

5.11 SEND

Conforming implementations accept and process SEND commands. With each SEND command during normal data transfer, the interface sends a T-DATA request to transport. The user data for that request is the data contained in the SEND command's buffer preceded by the two octet NetBIOS header. (Note that the NetBIOS header is attached to datagram as well as connection oriented Virtual Circuit traffic.) If the interface has some knowledge of when the data is actually delivered to the user, it may withhold completion of the SEND until it knows of actual data delivery. If the interface has no such knowledge, it may complete the SEND at any time. The exact mechanism for determining when to complete the SEND command is a local matter.

If the NetBIOS interface has received a close request from the remote interface prior to receiving the SEND command from the local user, it accepts the SEND command but does not issue the T-DATA request. Since the data cannot be delivered to the remote user anyway, there is no need for the transport request. Of course, the interface also withholds completion of the SEND command until the close process completes. A SEND command retained in this manner is returned with an error code indicating that the session terminated.

5.12 CHAIN SEND

Conforming implementations accept and process CHAIN SEND commands. With each CHAIN SEND command, the interface sends a T-DATA request to transport. The user data for that request is the combination of both of the command's buffers, preceded by the two octet NetBIOS headers. If the interface has some knowledge of when the data is actually delivered to the user, it may withhold completion of the CHAIN SEND until it knows of actual data delivery. If the interface has no such knowledge, it may complete the CHAIN SEND at any time. The exact mechanism for determining when to complete the CHAIN SEND command is a local matter.

If the NetBIOS interface has received a close request from the remote interface prior to receiving the CHAIN SEND command from the local user, it accepts the CHAIN SEND command but does not issue the T-DATA request. Since the data cannot be delivered to the remote user anyway, there is no need for the transport request. Of course, the interface also withholds completion of the CHAIN SEND command until the close process completes. A CHAIN SEND command retained in this manner is returned with an error code indicating that the session terminated.

5.13 RECEIVE

Conforming implementations accept and process RECEIVE commands. When a user issues a RECEIVE command, the interface first looks for any user data

received for the ''session'' that has not yet been given to the user. If such user data exists, the interface copies the data into the RECEIVE command's buffer and completes the command. If the user data copied was the last of a T-DATA indication, the command completes successfully. If data still remains from the indication, the RECEIVE completes with a ''message incomplete'' (06H) return code.

If there is no data to satisfy the RECEIVE command, the interface simply keeps the command until data arrives or a time-out occurs. The RECEIVE may also complete if it is cancelled, if the ''session'' is closed. A RECEIVE command is not completed as a result of the local name being deleted.

5.14 RECEIVE ANY

Conforming implementations accept and process RECEIVE ANY commands. When a user issues a RECEIVE ANY command, the interface first looks for any user data received for an appropriate ''session'' that has not yet been given to the user (see ''T-DATA indication'' below). If such user data exists, the interface copies the data into the RECEIVE ANY command's buffer and completes the command. If the user data copied was the last data in a message, the command completes successfully. If data still remains to be delivered the RECEIVE ANY completes with a ''message incomplete'' (06H) return code.

If there is no data to satisfy the RECEIVE ANY command, the interface simply keeps the command until data arrives or a time-out occurs. The RECEIVE ANY may also complete if it is cancelled or if the local name is deleted.

5.15 SESSION STATUS

Conforming implementations must accept and process SESSION STATUS commands according to the NetBIOS definition. The field referred to as ''state of the session'' is not identical to the state of the NetBIOS/TP4 mapping described in Appendix I. The correspondence between the value returned by SESSION STATUS and the mapping state is:

Value returned in SESSION STATUS command	State of NetBIOS/TP4 mapping from Appendix I
IDLE (00H)	STA 00
LISTEN pending (01H)	STA 01
CALL pending (02H)	STA 02
Session established (03H)	STA 03, STA 05
HANG UP pending (04H)	STA 04, STA 08
HANG UP complete (05H)	STA 06
Session Ended Abnormally (06H)	STA 07

TABLE 3. Session Status Command Mapping

5.16 SEND DATAGRAM

Conforming implementations accept and process SEND DATAGRAM commands. When the implementation receives a SEND DATAGRAM, it first finds the transport address corresponding to the remote NetBIOS name. To find this address, it sends a resolve name request to the naming service module. If the naming services cannot resolve the name's address, the interface simply completes the SEND DATAGRAM command with an unsuccessful response code.

If naming services successfully resolves the remote name, and that name is a unique name, the NetBIOS interface sends a T-UNITDATA request with an appropriate destination transport address. That address consists of the NSAP address of the node on which the name resides, along with a T- Selector equal to the remote name. The interface then completes the SEND DATAGRAM command.

If the remote name is a group name, the interface also sends a T-UNITDATA request. In this case, however, the connectionless transport protocol will use the special multicast NSAP, and it will direct the datagram to the NetBIOS multicast MAC address and LLC service access point (see ''Broadcast Datagrams and Datagrams to Group Names'' in section 4.4). The datagram is not directed to a specific NSAP address of a node owning the group name. As with unique names, the destination T-Selector is equal to the remote name. After sending the T-UNITDATA request, the interface completes the SEND DATAGRAM command successfully.

5.17 SEND BROADCAST DATAGRAM

Conforming implementations must also accept and process SEND BROADCAST DATAGRAM commands. Since a SEND BROADCAST command does not specify a destination name, there is no need for name resolution. The interface simply sends a T-UNITDATA request to transport services with the special broadcast T-Selector for the destination T-Selector. The connectionless transport protocol will use the multicast NSAP, and it will direct the datagram to the NetBIOS multicast MAC address and LLC service access point (see ''Broadcast Datagrams and Datagrams to Group Names'' in section four above). After sending the T-UNITDATA request, the interface completes the SEND BROADCAST DATAGRAM command successfully.

5.18 RECEIVE DATAGRAM

Conforming implementations must accept and process RECEIVE DATAGRAM commands. When the interface receives a RECEIVE DATAGRAM command, it holds the command until an incoming datagram satisfies the command, the command is cancelled, or the local name is deleted. ''T-UNITDATA indication'' in the following section describes the actions the interface takes to successfully complete a RECEIVE DATAGRAM command.

5.19 RECEIVE BROADCAST DATAGRAM

Conforming implementations must accept and process RECEIVE BROADCAST DATAGRAM commands. When the interface receives a RECEIVE BROADCAST DATAGRAM command, it holds the command until an incoming datagram satisfies the command, or the command is cancelled. The command is also completed if the name is deleted. ''T-UNITDATA indication'' in the following section describes the actions the interface takes to successfully complete a RECEIVE BROADCAST DATAGRAM command.

6 TRANSPORT SERVICE INDICATIONS AND CONFIRMATIONS

In addition to generating appropriate transport service requests and responses, the NetBIOS interface must also respond appropriately to incoming transport service indications and confirmations. This section describes the responses to all of these service primitives.

In many implementations, the ISO transport services support upper layers other than the NetBIOS interface. Some transport service implementations, for example, may support both the NetBIOS interface and the ISO session protocol. This specification does not address the complications multiple upper layers introduce, and the primitives discussed below are assumed to be intended solely for the NetBIOS interface. For example, there is no attempt to describe how transport services know to pass a T-CONNECT indication to NetBIOS instead of to the ISO session services.

6.1 T-CONNECT Indication

When the NetBIOS interface receives a T-CONNECT indication, it looks for a pending LISTEN command to match the indication. A matching LISTEN command

must have a local name equal to the called T- Selector, and it must either have a remote name equal to the calling T-Selector or an unspecified (wildcard) remote name. If both a specific LISTEN and a wildcard LISTEN match, the specific LISTEN takes precedence.

If the interface matches a pending LISTEN command, it completes the command successfully and sends transport a T- CONNECT response. If no matching LISTEN exists, the interface sends transport a T-DISCONNECT request.

6.2 T-CONNECT Confirmation

When the NetBIOS interface receives a T-CONNECT confirmation, it completes the appropriate CALL command successfully.

6.3 T-DISCONNECT Indication

The actions the NetBIOS interface takes when it receives a T-DISCONNECT indication depend on the state of the affected ''session''. If that ''session'' has a CALL pending, the CALL command is completed with a ''session open rejected'' (12H) or a ''no answer (cannot find name called)'' (14H) return code. Which return code is returned depends on the reason given in the T-DISCONNECT indication. If the reason indicates that the remote TS user invoked the disconnect, the interface returns the call with a ''reject''ed return code; otherwise, it uses the ''no answer'' return code.

If the ''session'' is established when the T-DISCONNECT indication arrives, the interface completes any pending commands with the ''session ended abnormally'' (18H) return code. The interface also takes this action if the ''session'' is in the process of hanging up.

The only time an interface expects to receive a T-DISCONNECT indication is after sending a close response. In this case, the interface completes all pending commands with a ''session closed'' (0AH) return code. Additionally, if any RECEIVE ANY commands apply to the ''session'', one of those commands is also completed with ''session closed''. If no commands are pending on the ''session'', the interface waits for the user to issue another command. When the user issues a command, that command is completed with a ''session closed'' return code.

6.4 T-DATA Indication

A T-DATA indication tells the NetBIOS interface that data, a close request or a close response has arrived for a ''session''.

When the interface receives such an indication during normal data flow, it looks for a pending RECEIVE command with which to pass the data on to the user. If no RECEIVE command for the ''session'' is available, the interface looks for a pending RECEIVE ANY for the ''session's'' local name. If none are found, the interface then looks for a pending RECEIVE ANY for an unspecified (wildcard) name.

If the interface finds any command to satisfy the T-DATA indication, it copies the data into the command's buffer and completes the command. If all of the user data from the indication fits in the buffer, the command is completed successfully. If only part of the user data fits in the buffer specified by the command, the interface returns the command with a ''message incomplete'' (06H) return code. The interface then looks for another pending RECEIVE or RECEIVE ANY command in which to place the remaining data. The interface continues in this fashion until all of the data has been given to the user or until it can no longer find suitable commands.

If the interface cannot find a pending RECEIVE or RECEIVE ANY command, it keeps whatever user data is left until the user issues an appropriate

command.

If the NetBIOS interface receives a T-DATA indication, with a normal data NetBIOS header, after it has received a HANG UP command from the local user but before that HANG UP has completed, the T-DATA indication is simply ignored and the data discarded.

6.5 T-UNITDATA Indication

T-UNITDATA indications contain incoming NetBIOS datagrams. When the NetBIOS interface receives a T-UNITDATA indication, it examines the destination T-Selector to determine if the datagram is a broadcast datagram or if it is addressed to a specific name (see ''Broadcast Datagrams and Datagrams to Group Names'' in section four above).

If the received datagram is a broadcast datagram, the interface looks for pending RECEIVE BROADCAST DATAGRAM commands. If none exist, the interface discards the T- UNITDATA indication. If an appropriate NetBIOS command does exist, the interface copies the data from the T-UNITDATA indication to the command's buffer. If all the data fits in the buffer, the interface returns the RECEIVE BROADCAST DATAGRAM command with a successful completion. If the data exceeds the size of the buffer, the interface returns the command with a ''message incomplete'' (06H) return code, and the remaining data is lost.

If the received datagram is directed to a specific name, whether that name is a group name or a unique name, the NetBIOS interface ensures that the destination name is registered on its adapter. If the name does not exist on the local adapter, the interface discards the T-UNITDATA indication.

If the specific name exists on the local adapter, the interface searches for a pending RECEIVE DATAGRAM command for that name. If none exists, the interface then looks for a pending RECEIVE DATAGRAM command with an unspecified (wildcard) local name. If the interface is still unsuccessful, it discards the T-UNITDATA indication.

If an appropriate pending NetBIOS command does exist, the interface copies the data from the T-UNITDATA indication to the command's buffer. If all the data fits in the buffer, the interface returns the RECEIVE DATAGRAM command with a successful completion. If the data exceeds the size of the buffer, the interface returns the command with a ''message incomplete'' (06H) return code and the remaining data is lost.

6.6 T-EXPEDITED Data

This option is negotiated in the transport call request PDU as described in the MAP/TOP v3.0 specification. NetBIOS itself does not use Expedited Data, therefore T-EXPEDITED DATA Requests are never generated. If a T-EXPEDITED DATA indication is received, it is ignored.

7 NetBIOS NAME SERVICE PROTOCOL - OVERVIEW

This part, the remaining sections of this specification and Appendices II through V, defines a naming protocol for TOP networks that will support NetBIOS name support services.

7.1 Architecture

The NetBIOS Name Service is a distributed name service which provides facilities for naming objects in the internet environment, and for relating those names to useful attributes, such as protocol addresses.

The name service protocol provides a mapping of NetBIOS Names to their protocol (transport) addresses. The protocol is based on query/response primitives and a distributed information base. Every node on the network

maintains information regarding the services or names posted on that node. When a new name is to be added on any node, that node queries other nodes on the network to ensure that the name can be added. A similar process is followed to obtain the address of an object.

In a simple topology consisting of a few NetBIOS nodes on a broadcast based network, the name service protocol makes use of multicast addresses to register and resolve names. The name service element on NetBIOS nodes is called the NetBIOS Directory Service Element (NDSE). In a more complex topology having a large number of nodes, an internetworking environment or the presence of an OSI directory service, the use of a NetBIOS Directory User Agent (NDUA) is useful (but not required). If there exists an NDUA on the network, the NDSEs communicate with the NDUA using point-to-point datagram communications. NDUAs become the focal point of name service activity. NDUAs are expected to have the capability to interface with an OSI Directory User Agent (DUA) or interface with other NDUAs.

In the case when NDSEs cannot communicate with an NDUA, they revert back to multicast based communication among NDSEs. This limits the address resolution to the local subnetwork, since multicasts are not transported across subnet boundaries.

Figures 3 and 4 provide an example of a simple network topology.

The scenarios presented in this subsection depict the network activities involved for various name service related actions for internetwork communications and call-back type applications.

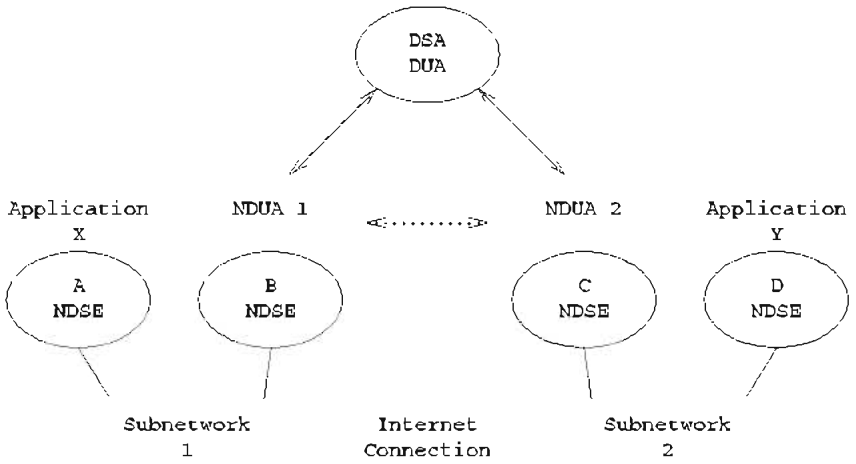


Figure 3. Name Service Example

NDSE Local NetBIOS Directory Service Entity, present on every node.

NDUA NetBIOS Directory User Agent, zero or more present on a subnetwork. At least one is needed for internet name service. It may also provide the interface to the ISO Directory Services (DUA-DSA), if present. It may also communicate with another NDUA using the name service protocol.

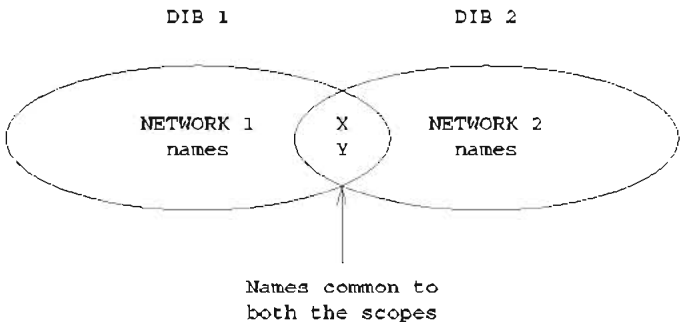


Figure 4. Name Scopes

The above topology, Figure 3, contains two subnetworks (1 and 2) with the associated NDUAs (NDUA1 and NDUA2 respectively). The following points identify the administrative actions of NDUAs to provide internetwork name resolutions.

- It is not possible for application programs using the NetBIOS interface to identify whether they wish to advertise in an internet environment. Therefore, NDUAs based on administrative filtering will update names in their directory information base (DIB) using DSA/DUA when the application programs register or unregister. The administrative filter mechanism is a local matter. It is expected that the names registered in the DIBs will be of ''server'' types providing services across internet boundaries.
- Application programs based on the call-back feature will also require administrative support. For example if the application X wishes to communicate with Y, and if it is necessary for both these applications to call each other, then the following steps can be taken by the respective NDUAs.
  - X will be posted on network 1 by application X, similarly Y will be posted on network 2 by application Y. Both these names will be entered in the DIB by their respective NDUAs.
  - Y will be posted by NDUA1 in the DIB with a pointer to the entry made by NDUA2. Similarly, X will be posted by NDUA2 in the DIB with a pointer to the entry made by NDUA1. This will serve the purpose of determining the uniqueness of ''globally'' known names within the scopes in which they are referenced.
  - If X & Y are unique names, then no other application can claim either of these two names in the two networks and associated DIBs, see Figure 4.
- Note that the information provided by the name service, particularly when using NDUAs will be ''loosely consistent'' in the sense that it may not be absolutely current.

7.2 High Level Feature Descriptions

The following set of features are provided by the NetBIOS Name Services. Some of these features are specifically developed for the NetBIOS environment, and for internetworking and performance reasons. A brief and high level description of each of the features follow.

- NetBIOS: The name service supports a flat, NetBIOS compatible name space. Names need be unique only within the context of the local subnet.

- Standards: The name service requires minimum functionality from underlying layers, a simple standard datagram transfer service is all that is needed. Also, name service is architected with migration to the ISO directory service in mind. A deliberate effort is made to ensure that we provide ISO compatible name services in a way that allow a smooth transition to a ''real'' ISO directory service when it is fully specified.
- Internetworking: The name services provide support for internetwork communication. Access to the name service is transparent to the application programs. Internet name resolution is supported. All intranet name resolution is supported by the distributed database, multicast, or point-to-point mechanisms. The name service is integrated with ISO transport service to allow the exchange of information relative to transit delay associated with a particular resource (e.g. 1200 baud link). Transit delay information is important to allow support of NetBIOS applications with dependencies on Receive-Time-Out or Send-Time-Out (RTO/STO).
- Graceful Degradation: Loss of a single node affects only local calls to that node. Loss of a NetBIOS Directory Service Entity (NDSE) on a node affects only local calls to that node. Loss of an NDUA affects only internet name resolution. Name resolution continues after the loss of an NDUA by using the multicast operation mode of the name service.
- Remote Adapter Status: The name service is integrated with support for Remote Adapter Status. A user can issue a status request on a NetBIOS name and will receive the status information associated with the node on which that end point exists, even if the node is on another subnetwork. Note that additional information regarding complete use of this service is provided in Appendix III.
- Compatibility: The NetBIOS names are used for T- Selectors (transport service access point identifiers.) This provides a simple, efficient and effective mapping between NetBIOS names and T-Selectors which becomes a part of the transport address (t-selector+nsap address with null ssap and null psap). NetBIOS is implemented on ISO Transport Class 4 (8073) and ISO Connectionless Transport (8602). Thus, NetBIOS based products and other TOP applications can coexist on the same network and on the same node.
- Set of Functions: A set of functions are defined. The name service makes use of three types of messages, request/advise, response and pending. Names, or objects, are associated with a set of attributes which include, among other things, full transport address (with null psel and null ssel) of the object.

  The set of functions supported are:

  a. Register Name

  b. Register Group Name

  c. Adapter Status

  d. Unregister Name

  e. Resolve Name

  f. Advise Name Conflict (Generation and Response)

  g. Advise NDUA Present

7.3 Scope and Purpose

This specification presents the NetBIOS Name Service Protocol (NSP). The NSP is the basic transfer mechanism for exchanging name service requests between systems. The NSP mechanism and protocol is specified here to support the needs of the NetBIOS Name Service. It is currently used only by the NetBIOS

directories, but it is constructed to allow for expansion to other directory applications.

It consists of high-level operations that support name registration, resolution and attribute association.

7.4 Underlying Services

The NetBIOS Name Service Protocol is based on datagram services provided by CLTP (see Figure 2) with a maximum TPDU size of 1024 octets.

7.5 NetBIOS Name Service (NS)

Operations supported by the NS include name registration and resolution, the storage, and the deletion of attribute information associated with names. These operations were conceived with the ISO/CCITT Directory Services model in mind, and should ease migration to that environment.

The following background information is useful when reviewing the protocol:

- the name of an object (usually an application entity) can be thought of as a search key for retrieving information about the object;
- information takes the form of attributes which describe the characteristics of an object (such as its protocol address);
- the distributed directory database maintains this information in records known as attribute tuples, which are encoded in a Type-Length-Value format.

7.6 Services

The NetBIOS Name Service Protocol primitives are summarized in Table 4:

Primitives		Parameters
NB_RegisterName	.Request/ .Indication	NB_Name, NB_InitialAttributesList
	.Response/ .Confirm	NB_ResponseCode
NB_RegisterGroupName	.Request/ .Indication	NB_Name, NB_InitialAttributesList
	.Response/ .Confirm	NB_ResponseCode
NB_UnregisterName	.Request/ .Indication	NB_Name
	.Response/ .Confirm	NB_ResponseCode
NB_ResolveName	.Request/ .Indication	NB_Name, NB_RequestAttributesList
	.Response/ .Confirm	NB_ResponseCode, NB_Name, NB_ReturnedAttributesList
NB_AdapterStatus	.Request/ .Indication	NB_Name
	.Response/ .Confirm	NB_ReturnedAttributesList
NB_NameConflictAdvise	.Request/ .Indication	NB_Name, NB_AdviseAttributeList
NB_NDUAHereAdvise	.Request/ .Indication	NB_InitialAttributeList

TABLE 4. Service Primitives for Name Service Protocol

8 NetBIOS NAME SERVICE PROTOCOL FUNCTIONS

8.1 General

This section describes the functions performed as part of the name service. All the functions described here are mandatory.

8.1.1 Response Semantics

The values given in the following sections for setting the Response-Semantics field in the name service PDUs serve as guidelines only.

Individual implementations may choose to use different values. However, any example given assumes the use of the recommended values.

8.1.2 Multicast Requests versus Requests to NDUA

In general, the operation of these functions will depend on the NDSE's reaction to the presence of an NDUA. When these functions issue remote requests, they operate as follows:

1. If an NDSE does not know the address of an NDUA, it proceeds to Step 2. Otherwise, the request is sent as a point-to-point datagram to the NDUA, as follows:

   a. DestinationAddress is set to the transport address of NDUA.

   b. ProcedureTimeout is set to ''T'' seconds. The value of ''T'', as well as the manner in which ''T'' may be configured, is left as a local matter.

   c. ResponseSemantics is set to Unconditional Response.

   d. Other portions of the request PDU are set as appropriate for each function. See below for details.

   e. The request is sent as a point-to-point datagram to the NDUA. If no response is received within ''T'' seconds, the request is retransmitted every ''T'' seconds until such time as a response is received or until some maximum number of retransmissions has been reached (see also section 8.7). The maximum number of times a given request may be sent to an NDUA is denoted by ''X'' (X>=1). The value of ''X'', as well as the manner in which ''X'' may be configured, is left as a local matter.

   f. If no response is received after ''X'' transmissions, proceed to Step 2. If a response is received, then the function will complete by sending either a success or failure indication to the originator depending on the response received, and Step 2 is not performed.

2. In the absence of an NDUA (or no response from NDUA after ''X'' tries), the request is sent as a multicast datagram to all other NDSEs, as follows:

   a. DestinationAddress is set to the transport address that represents ''ALL NetBIOS DIRECTORY SERVICE ENTITIES''. This address consists of the t-selector reserved for NDSEs and the multicast NSAP. See Appendix IV for details.

   b. ProcedureTimeout is set to ''T'' seconds. The value of ''T'', as well as the manner in which ''T'' may be configured, is left as a local matter.

   c. ResponseSemantics is set as recommended for each function. Details are given below for each function.

   d. Other portions of the request PDU are set as appropriate for each function. See below for details.

   e. The request is sent as multicast datagram to all NDSEs. If no response is received within ''T'' seconds, the request is retransmitted every ''T'' seconds until such time as a response is received or until some maximum number of retransmissions has been reached (see also Section 8.7). The maximum number of times a given request may be sent to NDSEs is denoted by ''Y'' (Y >= 1). The value of ''Y'', as well as the manner in which ''Y'' may be configured, is left as a local matter.

   f. If no response is received after ''Y'' transmissions, then the function will complete either a success or failure indication to the originator depending on the ResponseSemantics used. (If Response on Success was used, then failure is assumed. If Response on Failure was used, then success is assumed, etc.)

      If a response is received, then the function will complete by sending either a success or failure indication to the originator depending on the response received.

8.1.3 Actions of NDSE (or NDUA) on Receipt of Remote Request

In general, when an NDSE (or NDUA) receives a request PDU from other NDSEs or NDUAs it will process the request and return a response PDU as appropriate. The general actions of NDSE are as given below. More specific actions of NDUA are given in Appendix V.

1. All the response PDUs must contain the same source reference that was provided in the request PDU.
2. If for any reason, the NDSE expects a delay in processing the request within the ProcedureTimeout value provided in the request PDU, it must return a point-to-point pending PDU to the originator.
3. The NDSE must return the Response PDU based upon the type of request and the ResponseSemantics.
   a. A response PDU is always returned if Unconditional Response was requested.
   b. A response PDU is returned if the operation was a success (or a partial success) and Response on Success was requested.
   c. A response PDU is returned if the operation was a failure and Response on Failure was requested.

8.2 Register Name Function

This function is responsible for verifying the unambiguity of a new (non-group) name, registering the name on the network, and, optionally, associating attributes with the name.

Name service clients are allowed to choose a name for their application entities, but a name must be determined to be unambiguous; that is, not already in use[10]. The function queries all relevant databases, local or remote, to determine if the name is already in use. If the name is not found, the function assumes that the name is unclaimed and registration succeeds. If the name is found to already exist, the function aborts and returns a failure indication to the originator.

The following actions are taken by this function:

1. If the name exists in the local (node) version of the specified database, the entire procedure is aborted and a failure indication is returned; otherwise, the name is tentatively registered (put into ''being registered state'') in the local database in order to avoid race conditions with other systems adding the same name; and this name is defended by generating responses to the received Register Name Requests and Register Group Name Requests as if the name were registered, but will respond to the Resolve Name Request as if the name were not registered.
2. A request is sent to an NDUA or all NDSEs, as described in Section 8.1.2. Parameter values particular to the Register Name Request are set as follows:
   - Procedure is set to NB_RegisterName;
   - DestinationAddress is set to the transport address of a valid NDUA, otherwise to the transport group address that indicates ''ALL NetBIOS DIRECTORY ENTITIES'';

10. Note that this does not apply to group names which are ambiguous by definition. Group names are registered using the Register Group Name Function.

- ProcedureTimeout is set to ''T'' seconds;
- ResponseSemantic is set to Unconditional Response if NDUA[11] address is specified, otherwise it is set to Respond-on-Failure. Note that the NDSE trying to register a name will receive a response, success or failure if there exists an NDUA on the network. Otherwise it will receive a failure response with response code of Registration Error.
- NB_Name is taken from the original NB_RegisterName.Request;
- NB_Initial Attribute List contains at least two elements, i.e., protocol address and unique attribute.

3. If a failure response is received from any NDUA or NDSE, the name is already in use on another node. In this case, the tentative registration in the local database is cancelled, the procedure aborts, and a failure indication is returned to the originator.

   If a successful response is received from an NDUA (indicating either the name was unknown or the name was previously registered to the NDUA with the same protocol address as specified in the current request) or if no response is received from any NDSE, then the name is considered to be claimed by the local node. The tentative registration of the name in the local database is made permanent, and the procedure completes by sending a success indication to the originator.

4. The return code is returned in the NB_ResponseCode.

See Appendix II for a set of sample PDU encoding generated by a typical NB_RegisterName function.

8.3 Register Group Name Function

This function is responsible to verify the unambiguity of a new group name, registering the name on the network, and, optionally, associate attributes with the name.

Names on the network must normally be unique; that is, referring to only one owner. In the case of group names, however, the name is allowed to be shared by several owners so long as all the owners recognize the situation. This function is used when an application specifically wishes to share a name with other applications.

This function queries all relevant databases, local or remote, to determine if the name is already in use as a unique name. If a unique version of the name is not found, the function assumes that the name is free to be claimed as a group name, and registration succeeds. If the name is found to already exist in a unique form, the function aborts and returns a failure indication to the originator.

This function performs the following actions:

1. If a unique version of the name exists in the local version of the appropriate database, the entire procedure is aborted and a failure indication is returned; otherwise, the name is tentatively registered (put into ''being registered state'') in the local database in order to avoid race conditions with other systems adding the same name as a unique name. While the name is tentatively registered, this node will defend the name by generating responses to the Register Name Requests as if the name were actually registered, but will respond to Resolve Name Requests as if

11. The operation of NDUA and NetBIOS Object Class definition is given in Appendix V.

the name were not registered.

2. A request is sent to an NDUA and/or all NDSEs, as described in Section 8.1.2. Parameter values particular to the Register Group Name Request are set as follows:

   - Procedure is set to NB_RegisterGroupName;
   - ProcedureTimeout is set to ''T'';
   - ResponseSemantics is set to Unconditional Response if an NDUA address is specified, else it is set to Respond-on-Failure. Note that the NDSE trying to register a name will receive a response, success or failure, if an NDUA exists on the network. Otherwise it will receive a failure response with response code of Registration Error;
   - NB_Name is taken from the original NB_RegisterGroupName.Request;
   - NB_Initial Attribute List contains at least two elements, i.e., protocol address and group attribute.

3. If a failure response is received from any NDUA or NDSE, the name is already in use on another node as a unique name. In this case, the tentative registration in the local database is cancelled, the procedure aborts, and a failure indication is returned to the originator.

   If a successful response is received from an NDUA (indicating either the name was unknown or the name was previously registered to the NDUA as a group name) or if no response is received from any NDSE, then the name is considered to be claimed by the local node. The tentative registration of the name in the local database is made permanent, and the procedure completes by sending a success indication to the originator.

4. The return code is returned in the NB_ResponseCode.

See Appendix II for a set of sample PDU encodings generated by a typical NB_RegisterGroupName function.

8.4 Unregister Name Function

This function is used to remove a registered name from the network.

This function attempts to update or remove both local and remote database entries corresponding to this name. In the case of a unique name, all attributes associated with the name are deleted from the entry, and the name is released. In the case of a group name, specific sets of attributes contained in the Unregister Name Request (viz. transport address) are deleted, and the name is released when the last set of attributes are deleted.

Note that if the node just ''disappears'' without unregistering a name, it is possible that cached entries and NDUA databases may contain invalid entries. The name service is designed to be ''loosely consistent'' and allows for the possibility of invalid entries, so the protocol will still function when a node ''disappears''.

This function performs the following actions:

1. If the name does not exist in the local (node) version of the appropriate database, the entire procedure is aborted and a failure indication is returned.

2. A request is sent to an NDUA and/or all NDSEs, as described in Section 8.1.2. Parameter values particular to the Unregister Name Request are set as follows:

- procedure is set to NB_UnregisterName;
- if the name is being unregistered in other domains (scopes) or Directory Service Agents (DSAs) then for every DSA an Object Identifier is included in the request;
- ResponseSemantics is set to Unconditional Response if NDUA is specified, or else it is set to No Response. In addition, when NDUA receives such a request it re-multicasts this request on the local subnetwork;
- NB_Name is taken from the original NB_UnregisterName Request;
- NB_InitialAttributeList contains at least one element, viz., the protocol (transport) address associated with the name.

3. The return code is in the NB_ResponseCode.

8.5 Resolve Name Function

This function is used to resolve a name to a set of attributes (most commonly a Transport Address). If such an entry exists in a local or remote database, the requested attributes are returned to the originator along with a success indication. If the entry is found but not all requested attributes are known, then those attributes which are known and requested are returned along with a partial-success indication. If no such entry can be found, the procedure returns a failure indication to the originator.

The following actions are taken by this function:

1. A request is sent to an NDUA and/or all NDSEs, as described in Section 8.1.2. Parameter values particular to the Resolve Name Request are set as follows:
   - Procedure is set to NB_ResolveName;
   - ProcedureTimeout is set to ''T'' seconds;
   - ResponseSemantics is set to unconditional response if an NDUA address is specified, otherwise it is set to Respond-on-Success;
   - Arguments for the remote NB_Resolvename procedure, if NDUA is specified, are as specified below.
     - NB_Name is taken from the original NB_ResolveName.Request;
     - NB_RequestAttributesList is taken from the same parameter on the original NB_ResolveName.Request.
2. If a failure response is received from any NDUA or if the request(s) to NDSEs timed out without response, then the name is unknown. In this case, the procedure aborts and a failure indication is returned to the originator.
   - It is possible that the resolve name response may contain fewer attributes than requested. In such a case, the response code will be of partial success. Such responses are also treated as a ''successful response''.

   If a successful response is received from an NDUA or an NDSE, then the requested, or received attributes, when fewer attributes are received, are returned to the originator with an indication of success.
3. The return code, name and requested attributes are returned as the NB_ResponseCode, NB_Name and NB_ReturnedAttributesList parameters, respectively, with the above parameters being passed as NB_ReturnedAttributesList.

A successful resolve name response must have the requested transport address attributes. It is possible that, if the resolve name response is received from NDUA it may contain more than one transport address when the name is a group name. Similarly, resolve name responses may come from several NDSEs when the name is a group name. Also, note that it is possible all the attributes may not fit in a PDU. In that case the attribute list is truncated based on local choice.

See Appendix II for a set of sample PDU encodings generated by typical NB_ResolveName functions.

8.6 Name Conflict Advise Function

This function consists of two parts. The first part of the function requires detection of conflict, and the second part requires the processing of the ''NameConflictAdvise'' indication.

This function is used to detect names in ''conflict''. It is possible, though by remote chance, that a given subnetwork will contain two or more identical unique names, or one or more identical group names along with at least one identical unique name posted in the name service databases, such that every node posting such name thinks that it has posted a unique name.

The function is defined in two parts. The first part is associated with the detection of conflict. It requires that the node resolving a name detects more than one response to a resolve name request (either by waiting for or by accepting more than one response.) If more than one response is received, for a unique name, it indicates that the name is in conflict. The node detecting the conflict sends a point-to-point advise (NameConflictAdvise PDU) back to all but one (generally the first) responder indicating that that name posted is in conflict.

The second part of the function is associated with the processing of a ''NameConflictAdvise'' indication. When a node receives the conflict indication, it will set the ''Name-In-Conflict'' attribute for that name. When all the current sessions are terminated that are associated with a name with the ''Name-In-Conflict'' attribute set, the name should be removed/unbound/deleted from its database by explicit user delete name command. During this period, the node will not allow the use of that name for any other ACTIVITY other than for currently active sessions and adapter status.

8.7 Pending Function

This Pending function is invoked by the receiver of a request PDU if it expects a longer delay in processing the request than the procedure timeout indicated in the request PDU. The response PDU is returned to the source of the request with the type field set to ''pending'' and the procedure timeout field set to a new timeout value.

8.8 NDUA Here Advise Function

This function generates the ''NDUA here PDU'' to announce the presence of an NDUA on a subnetwork. This function is used only by NDUAs. An NDUA uses this function to multicast a message when it first joins the subnetwork. It also uses the function to send point-to-point messages to NDSEs which may be unaware of an NDUA's presence. See Appendix V for further details.

8.9 Special Comments

8.9.1 Cache

Cache table cleanup may be a concern in various applications. However, the mechanism chosen to cleanup the cache table may or may not be desirable, depending on a particular application. This protocol does not provide any indication when a name is unadvertised, because there can be no guarantee that such an indication will always be given.

It is possible to associate timers with every name in the cache table, so that names are deleted after a finite amount of time. In addition, it also possible to send ''keep- alive'' PDUs periodically for every posted name. However, both these techniques become cumbersome for a large network or network with many posted names. Therefore, maintaining a cache is treated as a local matter. Caches are set-up for reasons of performance. The protocols do not specify or recommend a mechanism to maintain caches.

9 STRUCTURE AND ENCODING OF PDUs

9.1 Structure

All the Protocol Data Units shall contain an integral number of octets. The octets in a PDU are numbered starting from 1 and increasing in the order they are put into a TSDU. The bits in an octet are numbered from 1 to 8, where bit 1 is the low-order bit. Note that the name service PDUs do not carry the two octet NetBIOS Header.

When consecutive octets are used to represent a binary number, the lower octet number has the most significant value.

When the encoding of a PDU is represented using a diagram in this section, the following representation is used:

1. octets are shown with the lowest numbered octet to the left, and higher number octets to the right;
2. within an octet, bits are shown with bit 8 to the left and bit 1 (least significant) to the right.

PDUs shall contain, in the following order:

1. the fixed part;
2. the variable part.

9.2 Fixed Part

9.2.1 General

The fixed part contains frequently occurring parameters such as the PDU type and total length.

If any of the parameters of the fixed part have an invalid value, it constitutes a protocol error and the offending PDU shall be discarded.

The format of the fixed part is shown in Figure 5.

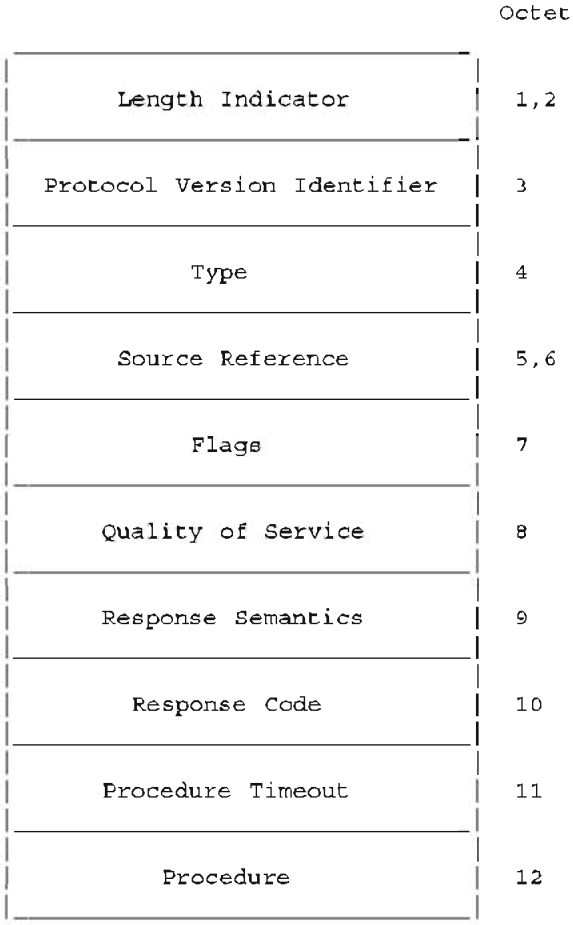


Figure 5. PDU Header - Fixed Part

9.2.2 Length Indicator

This field is contained in octets 1 and 2 of the PDU. The length is indicated by an unsigned binary number, with a maximum value of 65534, and the value 65535 (1111 1111 1111 1111 or -1) is reserved for future extensions. The length indicated shall be the header length in octets, but excluding the length indicator field.

Note that this protocol defines PDUs as consisting entirely of header, since there is no facility for carrying user data.

9.2.3 Protocol/Version Identifier

This field is contained in octet 3 of the PDU. The value of this field for the first release shall be 0001 0001.

PDUs containing protocol/version identifiers with different values shall be considered a protocol error.

9.2.4 Type

This field identifies the PDU type and is contained in octet 4. It is used to define the structure of the variable part of the PDU. Valid codes are given in Table 5.

Type	Binary Value
REQUEST pdu	0000 0010
RESPONSE pdu	0000 0100
PENDING pdu	0000 1000
ADVISE pdu	0001 0000

TABLE 5. Valid PDU Type Codes

All other values are reserved and shall constitute a protocol error.

9.2.5 Source Reference

This field is contained in octets 5 and 6. It identifies a specific invocation of a request and is used by the initiator to correlate responses with the appropriate requests. The value for this field is selected by the initiator and is returned (but not interpreted) by the responder. The same value is used in the successive retransmissions of the PDU.

9.2.6 FLAGS

This field is contained in octet 7.

Every bit in the octet signifies a flag. Only two flags are defined.

1. The NDUA Flag - the least significant bit (binary value 0000 0001). Since NDUAs must also monitor and respond to broadcast messages destined to all NDSEs, it is important to be able to distinguish which of those messages were sent by an NDUA and which ones were sent from an NDSE. NDUA sets this flag in all the PDUs it generates; NDSEs reset this flag in all the PDUs they generate.
2. The Internet Flag - the second least significant bit (binary value 0000 0010). This flag is set by NDUA in the response PDU if the object being requested is across the LAN boundary, otherwise the flag is reset. This flag is always reset in a request PDU[12].
3. Other values are reserved.

9.2.7 Quality of Service Field

This field is contained in octet 8.

When the value of this field is set to zero in the request PDU, the destination entity is requested to provide the ''fastest'' answer, e.g. an NDUA only checking its local table. When it is set to ''255'', the responder is expected to provide its best answer, e.g. an NDUA ignoring its local table and obtaining current information from NDSEs[13]. The responder, similarly, will set this field to zero or ''255'' based on the answer provided. No other intermediate values for this field are defined.

9.2.8 Response Semantics

This field is contained in octet 9 of the PDU. It is set by the initiator to define the circumstances under which the responder should send a RESPONSE PDU. Allowable values are given in Table 6, and the responder must adhere to the rules given below. This field has meaning only in the request PDUs; in

12. This flag is useful for End Systems in two cases, (1) for the selection of the proper NSAP address for group names, and (2) for the selection of proper timer values for connections.

13. The definition of best is rather subjective. It implies that the responder is requested to make the most thorough check, e.g not just looking at the cached value but to revalidate the cache.

response PDUs this field is copied from the request PDU.

Response Semantic	Binary Value
No Response	0000 0000
Response on Success	0000 0001
Response on Failure	0000 0010
Unconditional Response	0000 0011

TABLE 6. Valid Response Semantics

All other values are reserved and shall constitute a protocol error.

The following rules shall be observed by the responder:

No Response
: No response is expected.

Response-on-Success
: The responder shall send a **RESPONSE** PDU only if the requested operation resulted in success or partial success (i.e., response code of S-success or S-partialResults, see below).

Response-on-Failure
: The responder shall send a **RESPONSE** PDU only if the requested operation resulted in failure.

Unconditional-Response
: The responder shall always send a RESPONSE PDU to indicate the result of the requested operation.

9.2.9 Response Code

This field is contained in octet 10 of the PDU. This 1-octet field is used to indicate the outcome of a requested operation. The high-order bit indicates success (0xxx xxxx) or failure (1xxx xxxx), with the other bits encoded to represent reasons. Table 7 shows a summary of the valid response codes.

Response	Code
S-success	0000 0000
S-partialResults	0000 0001
E-protocolError	1000 0001
E-nameNotFound	1000 0010
E-noAccess	1000 0011
E-registrationError	1000 0100
E-registrationNameInConflict	1000 0101
E-foundNameInConflict	1000 0110

TABLE 7. Valid Response Codes

S-success
: The request has been successfully completed.

S-partialResults
: The request has been partially completed, e.g. if the request was made for 2 attributes only one was found and returned. Note that the responding entity must not ''make up'' a value for an attribute that it does not have.

E-protocolError
: The request PDU violates the protocol (during normal operation this error must not be generated, it is a diagnostic tool, e.g., it is used when improper function code is received).

E-nameNotFound
: The name in resolve name request is not found.

E-noAccess
: The resources cannot be accessed, e.g. security or database not accessible, or name not found.

E-registrationError
: The register name request has been denied due to an already existing unique name when registering a unique or group name, or an already existing group name when registering a unique name.

E-registrationNameInConflict
: The register name request has been denied due to already existing name/s in conflict.

E-foundNameInConflict
: The resolve name request failed as the name found is in conflict.

9.2.10 Procedure Timeout

This field is contained in octet 11 of the PDU. It is interpreted as an unsigned binary number with a maximum value of 255 (1111 1111). It specifies the number of seconds the originator will wait before timing out the procedure.

The timeout value of 0 is valid; it indicates infinity (no timeout).

9.2.11 Procedure

This field is contained in octet 12 of the PDU. It identifies the remote procedure to be performed, and defines the format of the variable portion of the PDU. Allowable values are given in Table 8.

Procedure	Binary Value
NS-RegisterName	0000 0001
NS-RegisterGroupName	0000 0010
NS-UnRegisterName	0000 0011
NS-ResolveName	0000 0100
NS-AdapterStatus	0000 1000
NS-NDUAHereAdvise	0011 0000
NS-NameConflictAdvise	0010 0000
FUTURE DIRECTORY PROCEDURES	reserved

TABLE 8. Valid Procedure Codes

All other values are reserved and constitute a protocol error.

9.3 The Variable Part

9.3.1 General

The variable part is used to convey the parameters for the remote procedure, or values being returned from such a call. If the variable part is present, it may contain one or more parameters. Each remote procedure defines the number, type and order of parameters to appear in the variable part. The following are some of the most common parameters to appear in the variable part. Their order of appearance differs with the exact procedure call, and is defined in the PDU diagrams starting at sec. 9.5.

9.3.2 Name

This parameter is a variable length field used to unambiguously identify a database entry. It is usually set by the initiator and must be formed according to the rules for NetBIOS Names[14]. It is encoded in the format shown in Figure 6.

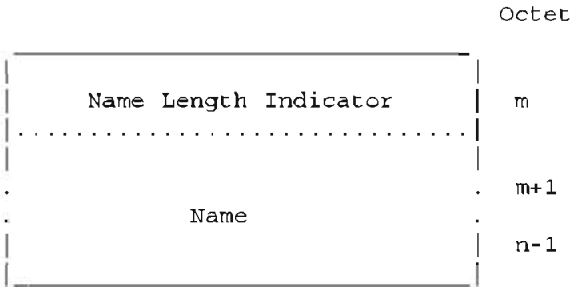


Figure 6. Encoding of the Name Parameter

9.3.3 Attribute Descriptor

This is a variable-length parameter which describes an attribute. Attribute descriptors may be specified by either the initiator (as in the case of a NB_ResolveName REQUEST pdu), or by the responder (as in the case of a NB_ResolveName RESPONSE pdu).

Attribute tuples are encoded in a standard type-length-value format as shown in Figure 7.

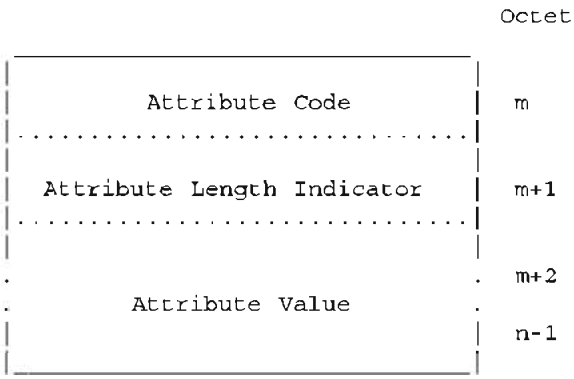


Figure 7. Encoding of an Attribute Descriptor

The Attribute Code field is a 1-octet binary value allowing a maximum of 254 different attribute types. The value of 255 is reserved for possible future extensions. The set of attribute codes in the range of 0-127 are reserved for TOP/NetBIOS use. The set of attribute codes in the range of 128-254 are assigned for private use (vendor specific). An implementation that does not recognize an attribute code will ignore the attribute. Table 9 lists the valid attribute codes defined by TOP/NetBIOS.

14. NetBIOS Names are defined to be consistent with the NetBIOS specifications to a length of exactly 16 octets.

Attribute	Attribute Value
Reserved	0000 0000
Reserved	1111 1111
Reserved	0000 0111 to 0111 1111
Unique Name	0000 0001
Transport Address	0000 0010
Name_In_Conflict	0000 0011
VC Accept	0000 0100
DG Accept	0000 0101
NodeAdminTransport Address	0000 0110
Private	1xxx xxxx*

* - values not including 1111 1111

TABLE 9. Disposition of Attribute Codes

An attribute (code) that is not recognized will be ignored. However, an unrecognizable attribute doe11not cause the entire request to be ignored. Recognized[15] attributes will still be registered (in the case of Registered Name and Registered Group Name Requests) or returned with a response code S-partialResults (in the case of Resolve Name Requests).

The Attribute Length field is a 1-octet binary value which indicates the length, in octets, of the attribute value field. The value field may be up to 254 octets in length. The value of 255 is reserved for possible future extensions.

The Attribute Value field contains the value of the attribute identified in the attribute code field. Encoding formats for standard attributes are specified in sec. 9.4.

9.3.4 Attribute Lists

In many operations, a list of attribute descriptors may be passed as parameters or return values. When such a list appears, it is preceded by an Attribute Count parameter. This parameter is a 1-octet binary value indicating the number of attribute descriptors in the list (see the previous section for the format of attribute descriptors). The field allows for a maximum of 254 attribute descriptors in the list. Such lists may contain only one item. The value 255 is reserved for possible future extensions.

The format of an attribute list is given in Figure 8.

15. Valid attributes, including private attributes, are recognized, and a list of valid attributes codes are given in Table 9.

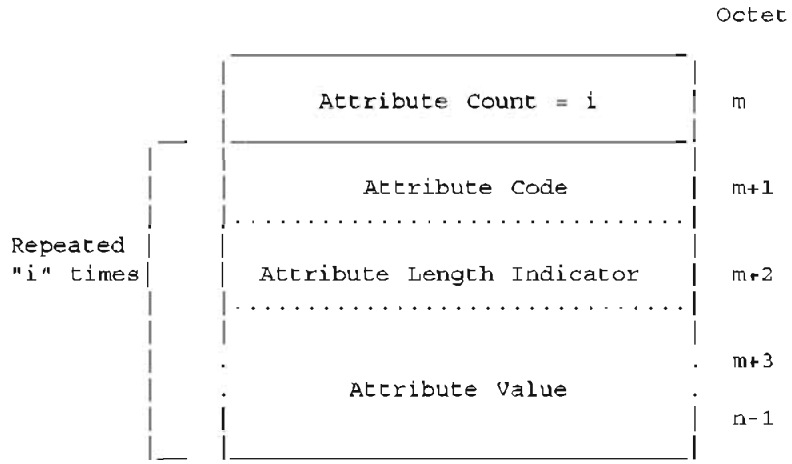


Figure 8. Encoding of an Attribute List

9.4 Encodings for Selected Attributes

9.4.1 General

When attribute tuples are passed in the protocol, they are encoded using a standard type-length- value format called an attribute descriptor (see sec. 9.3.3 for details). The following sections specify the contents of the Attribute Code, Attribute Length and Attribute Value fields for each of the standard attributes.

The following attributes are defined:

1. UniqueName
2. Transport Address
3. Name In Conflict
4. VC Accept
5. DG Accept
6. NodeAdminTransport Address

9.4.2 Encoding of the Attribute Code

In order to allow for new attributes to be added to the NetBIOS Name Service Protocol with a minimum of central coordination, the attribute code field is structured to represent a two-level hierarchy. The two levels are:

- attribute authority identifier (bit 8);
- attribute identifier (bits 1-7).

Attribute Authority Identifier

This field designates the authority responsible for allocating the attribute identifiers under its control. When the value of this field is set to zero (0), it indicates the value has been assigned by the TOP/NetBIOS SIG. The other values associated with this field set to one (1) indicate these are assigned locally for private use.

Attribute Identifier

This field designates the individual attribute within the domain of an attribute authority. Each attribute within a domain must have a unique

seven-bit code assigned by the reigning authority.

9.4.3 UniqueName

This attribute specifies whether the name corresponding to this entry is a unique name (as opposed to a group name).

Attribute Code: 0000 0001

Attribute Length: 1 octet

Attribute Value: Boolean (0xff=TRUE, 0x00=FALSE)

9.4.4 Transport Address

This attribute contains the Transport Address of the object. If this attribute is requested for a recognized name in a resolve name request, at least one transport address must be returned in the response. The encoding of the Transport Address attribute value field is as follows:

Attribute Code: 0000 0010

Attribute Length: variable

Attribute Value: See Figure 9

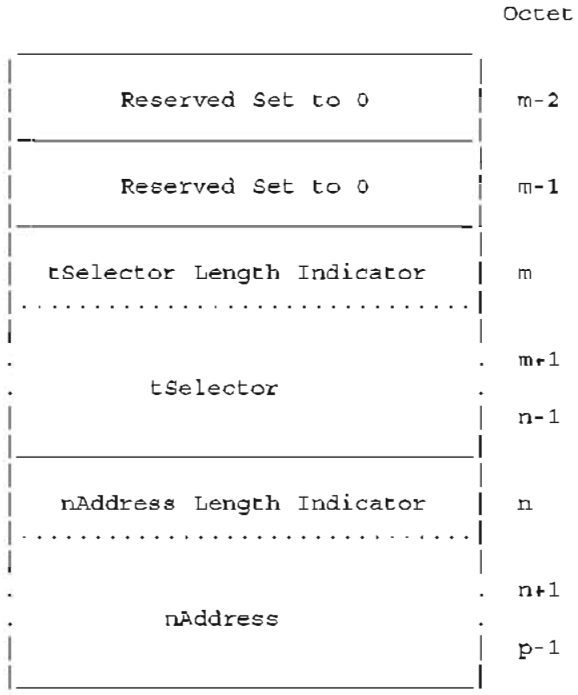


Figure 9. Value Field of Transport Address Attribute

9.4.5 Name In Conflict

This attribute indicates that the name is in conflict within its domain. Normally this attribute will be reset, when the name is added to the database. However, when it is detected that this name is in conflict this attribute is set. The name is said to be in conflict, when two or more objects with the same name and at least one of which with unique name attribute are present in the same domain[16].

Attribute Code: 0000 0011

Attribute Length: 1 octet

Attribute Value: Boolean (0xff=TRUE in conflict, 0x00=FALSE not in conflict)

9.4.6 VcAccept

This attribute specifies whether the server for this name is currently accepting VC connection requests, e.g., ''listen'' outstanding for that name. This attribute is only maintained by NDSEs. If these attributes are requested from NDUA then ''partial results'' may be returned[17].

Attribute Code: 0000 0100

Attribute Length: 1 octet

Attribute Value: Value (0x01-0xff=YES, 0x00=NO)

9.4.7 DgAccept

This attribute specifies whether the server for this name is currently accepting DG transactions, e.g. receive datagram/broadcast datagram outstanding for that name. This attribute is only maintained by NDSEs. If these attributes are requested from NDUA then ''partial results'' are returned.

Attribute Code: 0000 0101

Attribute Length: 1 octet

Attribute Value: Boolean (0xff=TRUE, 0x00=FALSE)

9.4.8 NodeAdminTransport Address

This attribute contains the Transport Address of the end-point used by Node Administration. This address is used for network management communication, e.g., for remote adapter status. The recommended address will be NDSE transport address. To obtain the ''remote adapter status'', the originating node will send out a query packet (Resolve Name Request) with this attribute set, and the responding node will return the address of the administrative entity (NDSE) on that node. The adapter status request is sent to this address. If this attribute is requested for a recognized name in a resolve name request, then this attribute must be returned in the response. The format of this attribute is the same as that of the ''transport address'' attribute.

Attribute Code: 0000 0110

Attribute Length: variable

Attribute Value: See Figure 9

9.5 PDUs for NB_RegisterName and NB_RegisterGroupName

---

16. Note that this attribute is not carried in any of the currently defined PDUs, but this attribute may be requested in a resolve name request, for administrative reasons. Internal implementation of this feature is a local matter for NDUAs and NDSEs. However, it is necessary to maintain this information locally.

17. The intent of the value for this attribute is to represent the number of VC requests the object is prepared to accept. A value of zero means the service is not available, and a value of 0xff means maximum service. It is a local matter to determine the current value of this attribute to be returned in the response PDU.

9.5.1 REQUEST PDU

The format of the REQUEST PDU is shown in Figure 10.

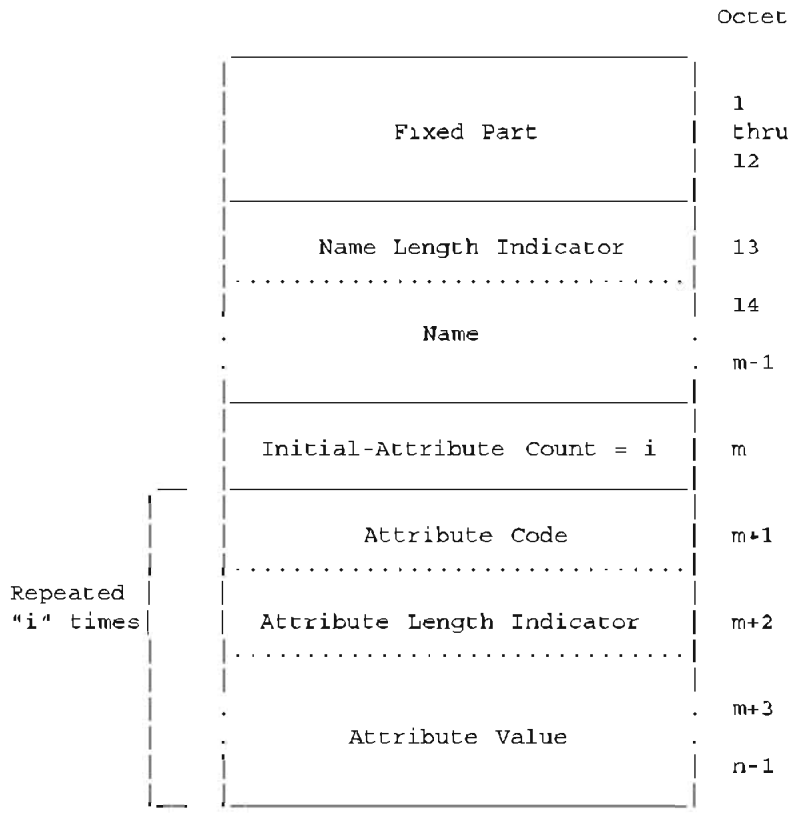


Figure 10. REQUEST PDU Format for NB_RegisterName and NB_RegisterGroupName

9.5.2 RESPONSE PDU

The format of the RESPONSE PDU is shown in Figure 11.

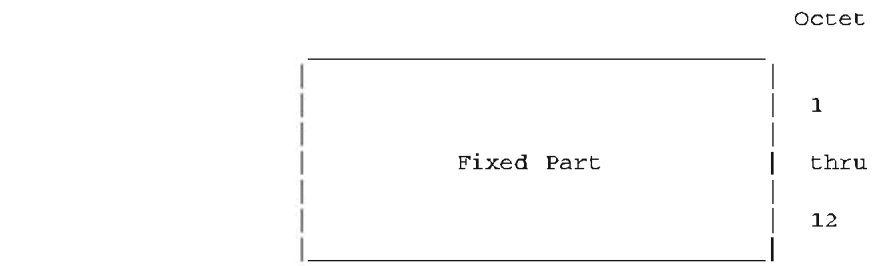


Figure 11. RESPONSE PDU Format for NB_RegisterName and NB_RegisterGroupName

9.6 PDUs for NB_UnregisterName

9.6.1 REQUEST PDU

The format of the REQUEST PDU is shown in Figure 12.

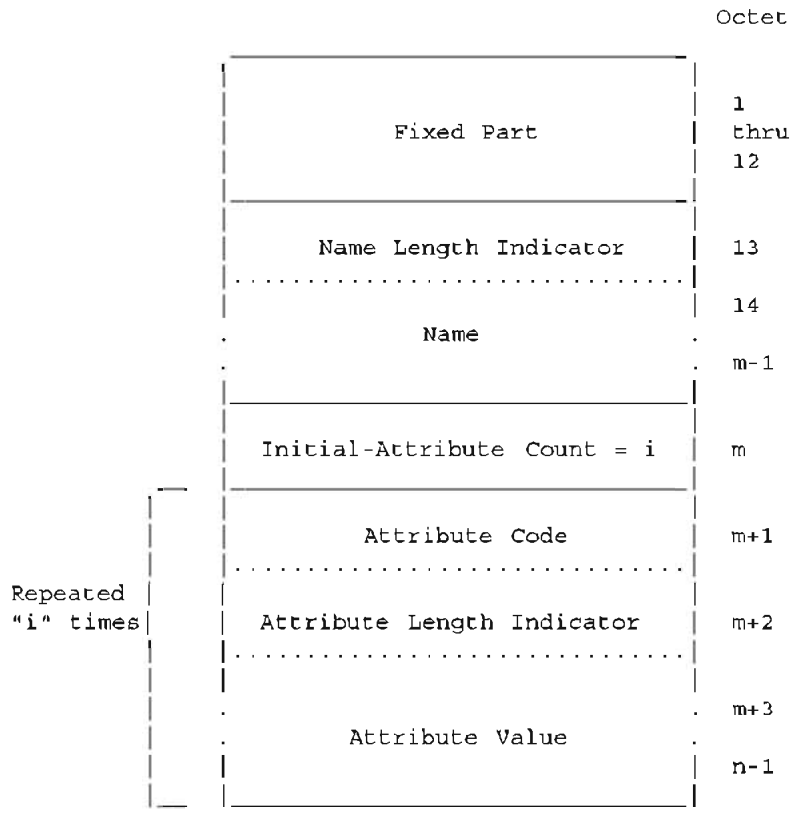


Figure 12. REQUEST PDU Format for NB_UnregisterName

9.6.2 RESPONSE PDU

The format of the RESPONSE PDU is shown in Figure 13.

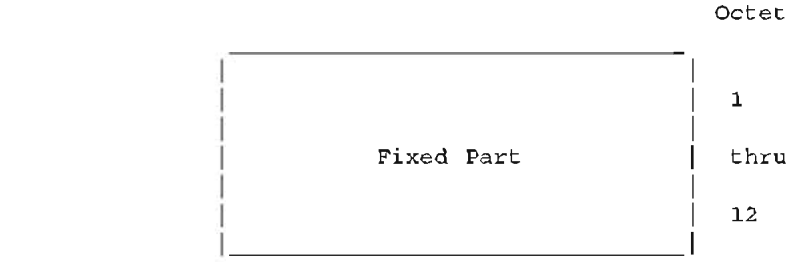


Figure 13. RESPONSE PDU Format for NB_UnregisterName

9.7 PDUs for NB_ResolveName

9.7.1 REQUEST PDU

The format of the REQUEST PDU is shown in Figure 14.

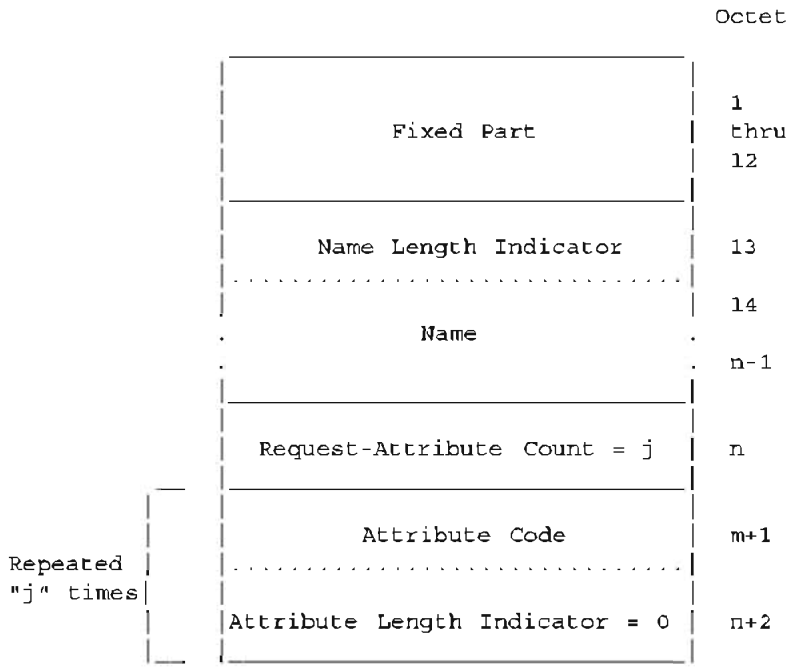


Figure 14. REQUEST PDU Format for NB_ResolveName

9.7.2 RESPONSE PDU

The format of the RESPONSE PDU is shown in Figure 15.

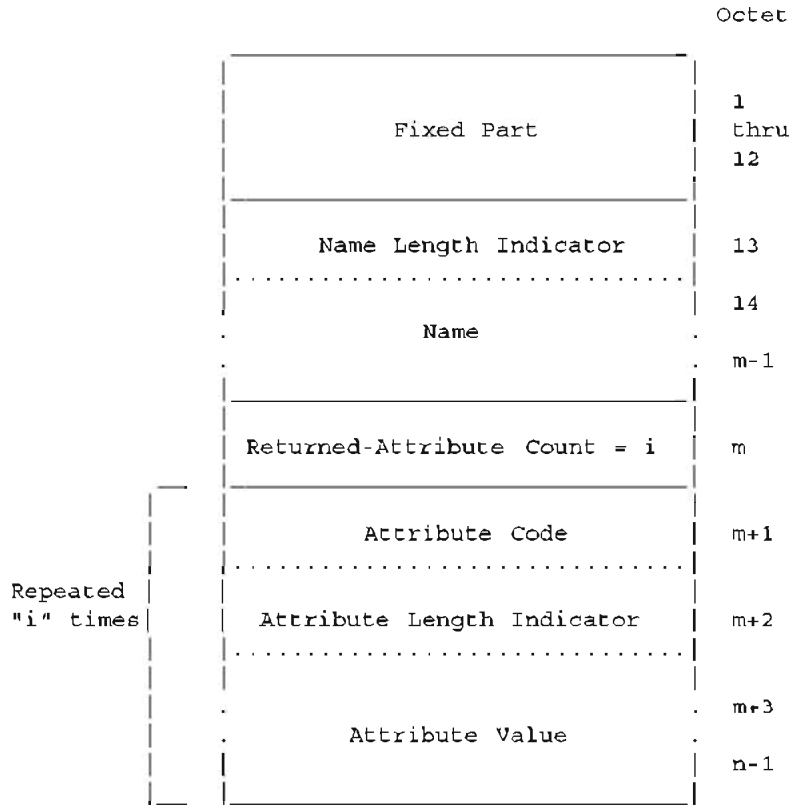


Figure 15. RESPONSE PDU Format for NB_ResolveName

Note that it is possible that the response PDU will contain fewer attributes than requested, but never more. Nodes must not make use of the source protocol control information (PCI) of a response to determine a name's address; they must parse the data contained in the response.

9.8 PDUs for NB_NameConflictAdvise

The format of the ADVISE PDU is shown in Figure 16.

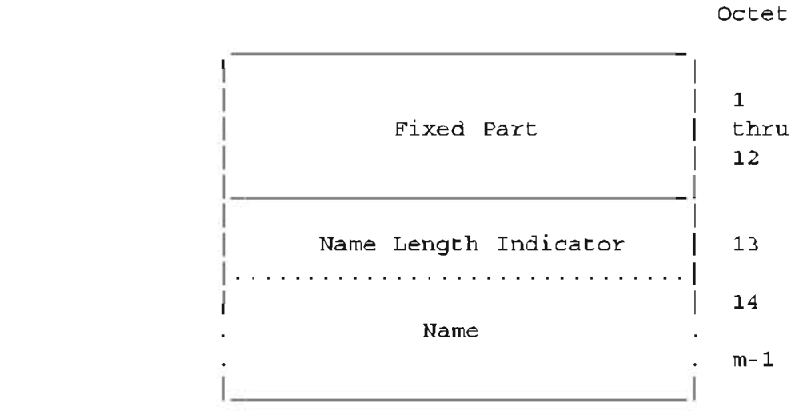


Figure 16. ADVISE PDU Format for NB_NameConflictAdvise

Note that the Type Code = ADVISE PDU Type.

9.9 PDU for NB_NDUAHere

The format for NB_NDUAHere, ''I am here'' PDU is given Figure 17.

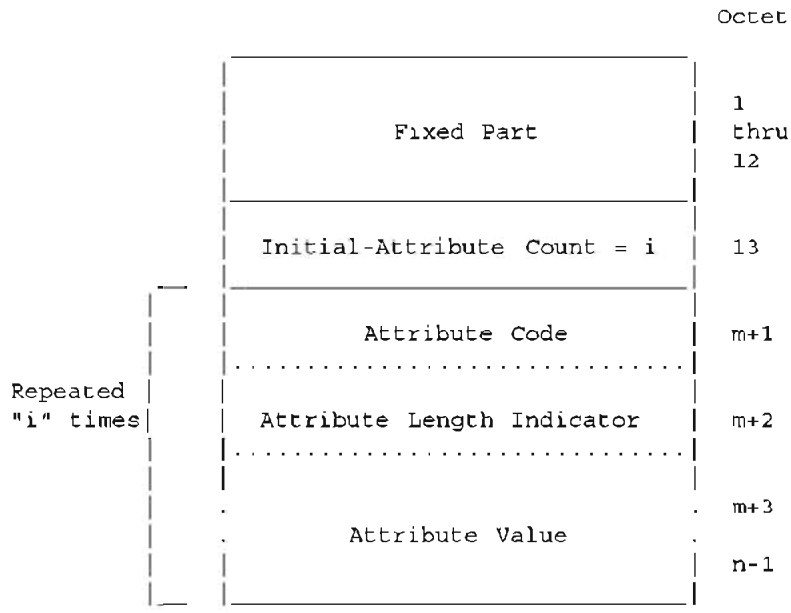


Figure 17. NDUA - I am here Advise PDU Format: NB_NDUAHere

Note that the Type Code = ADVISE PDU Type.

9.10 PDUs and Attributes

The intent of the following table is to provide general guidelines for the set of attributes that are ''meaningful'' with different PDU types. Note that Register means both unique and group registrations and address implies transport address. Attributes listed in square brackets imply optional. For example, the resolve name request may request for NodeAdmin Transport Address, or other attributes. The address attributes must be supplied in the response PDU when requested in a request PDU.

PDU Procedure	Request/Response	Attribute List
Register	Request	(name) Unique/Group Address
	Response	--
Unregister	Request	(name) Address
Resolve	Request	(name) Address [NodeAdmin-Address] [VC Accept] [DG Accept]
	Response	(name) Address(es) Unique/Group [VC Accept] [DG Accept] [NodeAdmin-Address]
NDUA Here	Advise	Address
Conflict	Advise	(name) Address

Figure 18. Sample PDUs and Attributes

APPENDIX I : STATE TABLES

This appendix is an integral part of the body of this specification. It presents, in an unambiguous form, the actions taken by the NetBIOS interface in response to user commands and transport primitives. The state tables detail the mapping between NetBIOS ''sessions'' and class four transport connections. They do not describe general, name service, or datagram service commands, nor do they attempt to show the interaction with NetBIOS name services. The state tables also omit any description of the validation procedures performed on each NetBIOS command; those procedures are adequately described in the NetBIOS interface definition.

The following subsections introduce the state tables by outlining the notation, conventions, actions and variables used by the tables. The tables themselves, which follow the text of this appendix, consist of six figures that specify the incoming events, states, outgoing events, specific actions, predicates and state tables. The actions defined by the state tables apply to a single NetBIOS ''session''. Each NetBIOS ''session'' operates under an independent state table.

I.1 Notation for State Tables

The state tables represent incoming events, states, and outgoing events with their abbreviated names. Tables 10, 11, and 12 specify these abbreviated names. The state tables represent specific actions with the notation [n], where ''n'' is the number of the specific action in Table 10. Predicates are represented by the notation pn, where ''n'' is the number of the predicate in Table 14. Notes are indicated by (n), where ''n'' is the note number at the foot of the figure. Finally, the tables show boolean operations with the characters ''&'' (logical and), ''|'' (logical or), and ''!'' (logical not).

I.2 Conventions for Entries in State Tables

The intersection of each state and incoming event in the state tables (Table 15) either is left blank, contains the notation ''//'', or contains an entry. If the intersection is blank, the incoming event is invalid. An invalid event can only occur if the NetBIOS interface commits an error. If the intersection contains ''//,'' it is logically impossible for the interface to receive the incoming event. Impossible events either cannot occur, or can only occur if an entity other than the NetBIOS interface (for example, the transport provider) commits an error. (These entries are often a consequence of the tabular presentation of the state tables.)

If the intersection of current state and incoming event contains an entry, the incoming event is valid and the entry specifies the actions the NetBIOS interface should take. Each valid entry either contains an action list or one or more conditional action lists. An action list may include outgoing events and specific actions, and it always specifies the resultant state. A conditional action list consists of a predicate expression made up of predicates and boolean operators, and an action list.

I.3 Actions to be Taken by the NetBIOS Interface

The NetBIOS interface takes the actions defined by the state tables (Table 15). Where those tables do not specify an action (if the incoming event is invalid or impossible), the action taken is a local matter.

For valid entries, if the intersection of the incoming event and state contains an action list, the NetBIOS interface takes the specific actions specified in the table. It then changes state to the indicated resultant state. If the intersection contains one or more conditional action lists, for each predicate expression that is true the NetBIOS interface takes the

specific actions in the order given by the action list for the predicate expression. If none of the predicate expressions are true, the incoming event is considered invalid and the actions taken are a local matter.

I.4 Variables

This specification defines several variables for the NetBIOS interface. The state tables use these variables to clarify the effect of certain actions and to clarify the conditions under which certain actions are valid. For purposes of this specification, these variables are purely logical entities; the way implementations actually represent them is a local matter.

- Nsto - timeout value for SEND and CHAIN SEND commands
- Nrto - timeout value for RECEIVE commands
- Vtca - False: the NetBIOS entity initiated the t- connect request (transport connection initiator), True: the NetBIOS entity received the t-connect indication (transport connection acceptor).

I.5 Incoming Events

Abbreviated Name	Name and Description
LISTEN	NetBIOS LISTEN command from user
CALL	NetBIOS CALL command from user
TCONind	T-CONNECT indication primitive
TCONcnf+	T-CONNECT confirmation (positive) primitive
TDATAind	T-DATA indication primitive
RECEIVE	NetBIOS RECEIVE or RECEIVE ANY command from user
SEND	NetBIOS SEND or CHAIN SEND command from user
SENDcnf	NetBIOS SEND or CHAIN SEND command confirmed
HANGUP	NetBIOS HANG UP command from user
CLSreq	Close request from remote interface
CLSrsp	Close response from remote interface
TDISCind	T-DISCONNECT indication primitive
STO	NetBIOS send timeout expiration
RTO	NetBIOS receive timeout expiration
TIM	Hang up timeout expiration

TABLE 10. Incoming Events

Notes:

The exact definition of SEND or CHAIN SEND command confirmation (see ''SENDcnf'' above) is a local matter. It is whatever event causes the interface to complete a SEND or CHAIN SEND command. Some implementations may define this event to be coincident with the SEND event; others may define it to occur when the buffer containing user data is returned to the NetBIOS interface, while still other implementations may define it to occur when the transport provider receives a transport level acknowledgement of receipt of the user data from the remote transport provider. Because the event cannot be precisely defined in this specification, the following state tables do not specify an implementation's actions when it receives a HANG UP command with SEND commands pending. Implementations are free to handle this case in any manner consistent with the NetBIOS definition and with this specification. Regardless of its exact definition, this event does not apply to the ''completion'' of close requests or close responses, despite the fact that they, like user data, are sent in TSDUs.

I.6 Outgoing Events

Abbreviated Name	Name and Description
TCONreq	T-CONNECT request primitive
TCONrsp+	T-CONNECT response (positive) primitive
LSTNcplt	Complete NetBIOS LISTEN command ''good''
CALLcplt	Complete NetBIOS CALL command ''good''
TDATAreq	T-DATA request primitive
SENDcplt	Complete NetBIOS SEND/CHAIN-SEND command ''good''
RCVcplt	Complete NetBIOS RECEIVE/RECEIVE-ANY command ''good''
CLSreq	Close request to remote interface
CLSrsp	Close response to remote interface
TDISCreq	T-DISCONNECT request primitive
HANGcplt	Complete NetBIOS HANG UP command ''good''

TABLE 11. Outgoing Events

Notes:

The completion of a NetBIOS command is only considered an outgoing event if the completion is successful, i.e., the command completes with a return code of ''good'' (0x00). This distinction, though somewhat arbitrary, does make the state tables more manageable.

I.7 States

Abbreviated Name	Name and Description
STA 00	Idle, ''session'' does not exist
STA 01	Listening
STA 02	Calling
STA 03	Established
STA 04	Hanging up, waiting for CLOSE RESPONSE
STA 05	Waiting for disconnect
STA 06	Closed, waiting to notify user
STA 07	Aborted, waiting to notify user
STA 08	Close Collison

TABLE 12. States

Notes:

For the correspondence between these states and the ''state of the session'' returned in the SESSION STATUS command, please refer to ''SESSION STATUS'' in section five.

I.8 Specific Actions

State	Description
[1]	set Nsto and Nrto to appropriate values
[2]	retain received data, waiting for RECEIVE or RECEIVE ANY command
[3]	discard data
[4]	return NetBIOS command with ''Command timed out'' (0x05) return code
[5]	return appropriate NetBIOS commands with ''Message incomplete'' (0x06) return code
[6]	return NetBIOS command with ''Session closed'' (0x0A) return code
[7]	return NetBIOS command with ''Session open rejected'' (0x12) return code
[8]	return all NetBIOS commands with ''Session ended abnormally'' (0x18) return code
[9]	terminate all pending RECEIVE commands and one RECEIVE ANY command with ''Session closed'' (0x0A) return code
[10]	start send timer
[11]	start receive timer
[12]	start hang up timer
[13]	cancel send timer
[14]	cancel receive timer
[15]	cancel all timers for ''session''
[16]	return NetBIOS command with ''No answer (cannot find name called)'' (0x14) return code
[17]	Set Vtca = false
[18]	Set Vtca = true

TABLE 13. Specific Actions

I.9 Predicates

p1	any RECEIVE or RECEIVE ANY commands available?
p2	enough RECEIVE or RECEIVE ANY commands available?
p3	more than one RECEIVE or RECEIVE ANY command required for the received data?
p4	retained data available for RECEIVE or RECEIVE ANY command?
p5	all of retained data from a single received TSDU fits in RECEIVE or RECEIVE ANY command?
p6	Any commands available to notify user of new ''session'' state?
p7	Does disconnect reason indicate ''remote TS user invoked''?
p8	Vtca = false ?
p9	Any command available, in addition to send or chainsend?

TABLE 14. Predicates

I.10 State Tables

STATE ----- EVENT	STA00 idle	STA01 listening	STA02 calling	STA03 established
LISTEN	[1] STA01	//	//	//
CALL	[1] [17] TCONreq STA02	//	//	//
TCONind	TDISCreq STA00	[18] TCONrsp+ LSTNcplt STA03	//	//
TCONcnf+	//	//	CALLcplt STA03	//
TDATAind	//	//	//	p1&p2&p3 [5] [14] RCVcplt STA03 p1&p2&!p3 [14] RCVcplt STA03 p1&!p2 [5] [2] [14] STA03 !p1 [2] STA03
RECEIVE	//	//	//	p4&p5 RCVcplt STA03 p4&!p5 [5] STA03 !p4 [11] STA03
SEND	//	//	//	[10] TDATAreq STA03
SENDcnf	//	//	//	[13] SENDcplt STA03
HANGUP	//	//	//	[12] [9] [14] CLSreq STA04

TABLE 15. State Tables

STATE ----- EVENT	STA00 idle	STA01 listening	STA02 calling	STA03 established
CLSreq	//	//	//	CLSrsp STA05
CLSrsp	//	//	//	
TDISCind	//	//	p7 [7] STA00 !p7 [16] STA00	p6 [8] [15] STA00 !p6 STA07
STO	//	//	//	p9 [4] [8] [15] TDISCreq STA00 !p9 TDISCreq STA07
RTO	//	//	//	[4] STA03
TIM	//	//	//	//

TABLE 15. State Tables (continued)

STATE ----- EVENT	STA04 wait close-resp.	STA05 wait disconnected	STA06 closed, waiting	STA07 aborted, waiting	STA08 close collision
LISTEN	//	//	//	//	
CALL	//	//	//	//	
TCONind	//	//	//	//	
TCONcnf+	//	//	//	//	
TDATAind	[3] STA04		//	//	
RECEIVE	[6] STA04	[11] STA05	[6] STA00	[8] STA00	
SEND	[6] STA04	[10] STA05	[6] STA00	[8] STA00	
SENDcnf		[13] SENDcplt STA05			
HANGUP	[6] STA04	[9] [15] HANGcplt STA05	[6] STA00	[8] STA00	
CLSreq	p8 CLSrsp STA04 !p8 STA08		//	//	//
CLSrsp	[15] HANGcplt TDISCreq STA00		//	//	CLSrsp STA5
TDISCind	[8] [15] STA00	p6 [6] [15] STA00 !p6 STA06	//	//	p6 [8] [15] STA00 !p6 STA07
STO	//	[4] [8] [15] TDISCreq STA00			
RTO	//	[4] STA05			
TIM	[8] [15] TDISCreq STA00	[8] [15] TDISCreq STA00			

TABLE 15. State Tables (end)

APPENDIX II : SAMPLE PDU ENCODINGS

II.1 Register Name Operation

The following tables contain sample PDU encodings for the NB_RegisterName REQUEST and RESPONSE PDUs exchanged as a result of a NB_RegisterName operation, with repeat count N=3 (X=1 and Y=2). In this example, the first request PDU is sent to NDUA, and the subsequent PDUs are multicasted assuming no response from NDUA.

Field	Value			
	PDU #1		PDU #2	PDU #3
Length Indicator	variable		same	same
Protocol/Version Indicator	0001 0001		same	same
Type	0000 0010		same	same
Source Reference	variable		same	same
FLAGs	reset		same	same
QOS	variable		same	same
Response Semantics	0000 0011	0000 0010	same	
Response Code	-		same	same
Procedure Timeout	variable		same	same
Procedure	0000 0001		same	same
Name LI	variable		same	same
Name	variable		same	same
Initial-Attb Count	n		n	n
List of Attributes	variable		same	same

TABLE 16. NB_RegisterName req. pdus generated by NB_RegisterName operation

The response PDU generated by the NDUA after successful registration of a name will have a Response Code of success. If an NDUA is not present on the network the response PDU will be generated by other Nodes with a Response Code of registration error if a name conflict exists.

Field	Value
	PDU #1
Length Indicator	0000 0000
	0000 1010
Protocol/Version Indicator	0001 0001
Type	0000 0100
Source Reference	variable
FLAGs	-
QOS	-
Response Semantics	0000 0011
Response Code	0000 0001
Procedure Timeout	variable
Procedure	0000 0001

TABLE 17. NB_RegisterName res. pdu generated by NDUA NB_RegisterName operation

II.2 Register Group Name

The following tables contain sample PDU encodings for the NB_RegisterGroupName REQUEST and RESPONSE PDUs exchanged as a result of an NB_RegisterGroupName operation, with repeat count N=3 (X=1, Y=2). In this example, the first request PDU is generated for NDUA and the subsequent PDUs are generated assuming no response from NDUA.

Field	Value		
	PDU #1	PDU #2	PDU #3
Length Indicator	variable	same	same
Protocol/Version Indicator	0001 0001	same	same
Type	0000 0010	same	same
Source Reference	variable	same	same
FLAGs	reset	same	same
QOS	variable	same	same
Response Semantics	0000 0011	0000 0010	same
Response Code	-	same	same
Procedure Timeout	variable	same	same
Procedure	0000 0010	same	same
Name LI	variable	same	same
Name	variable	same	same
Initial-Attb Count	variable	same	same
List of Attributes	variable	same	same

TABLE 18. NB_RegisterGroupName req. pdus generated by NB_RegisterGroupName operation

The response PDU generated by the NDUA after successful registration of name will have response code of success. If an NDUA is not present on the network the response PDU will be generated by other Nodes with Response Code of registration error if there exist a name conflict.

Field	Value
	PDU #1
Length Indicator	0000 0000 0000 1010
Protocol/Version Indicator	0001 0001
Type	0000 0100
Source Reference	variable
FLAGs	-
QOS	-
Response Semantics	0000 0001
Response Code	0000 0001
Procedure Timeout	variable
Procedure	0000 0010

TABLE 19. NB_RegisterGroupName res. pdu generated by NDUA NB_RegisterGroupName operation

II.3 Resolve Name

The following tables contain sample PDU encodings for the NB_ResolveName REQUEST, RESPONSE and PENDING PDUs exchanged as a result of an NB_ResolveName operation, with repeat count N=3 (X=1, Y = 2). In this example the first request PDU is generated for NDUA and the subsequent PDU are generated assuming no response from NDUA.

Field	Value		
	PDU #1	PDU #2	PDU #3
Length Indicator	variable	same	same
Protocol/Version Indicator	0001 0001	same	same
Type	0000 0010	same	same
Source Reference	variable	same	same
FLAGs	reset	same	same
QOS	variable	same	same
Response Semantics	0000 0011	0000 0001	same
Response Code	-	-	-
Procedure Timeout	variable	variable	same
Procedure	0000 0100	same	same
Name LI	variable	same	same
Name	variable	same	same
Request-Attb Count	0000 0010	same	same
Attb Code (UniqueName)	0000 0001	same	same
Attb LI	zero	same	same
Attb Code (TransportAddress)	0000 0010	same	same
Attb LI	zero	same	same

TABLE 20. NB_ResolveName req. pdus generated by NB_ResolveName operation

Field	Value
	PDU #1
Length Indicator	variable
Protocol/Version Indicator	0001 0001
Type	0000 0100
Source Reference	variable
FLAGs	-
QOS	-
Response Semantics	0000 0001
Procedure Timeout	variable
Procedure	0000 0100
Response Code	0000 0001
Name LI	variable
Name	variable
Returned-Attb Count	0000 0010
Attb Code (UniqueName)	0000 0001
Attb LI	0000 0001
Attb Value	1111 1111
Attb Code (TransportAddress)	0000 0010
Attb LI	variable
Attb Value	variable

TABLE 21. NB_ResolveName res. pdu generated by NB_ResolveName operation

The following points should be noted in the REQUEST and RESPONSE PDU encodings shown above:

- The UniqueName attribute in this example indicates that a system holding a unique version of the name is responding to the NB_ResolveName, although it could just have readily been a system with a group version of the name.

Field	Value
	PDU #1
Length Indicator	0000 0000
	0000 1010
Protocol/Version Indicator	0001 0001
Type	0000 1000
Source Reference	variable
FLAGs	-
QOS	-
Response Semantics	0000 0001
Procedure Timeout	variable
Procedure	0000 0100

TABLE 22. NB_ResolveName pending pdu generated by NB_ResolveName operation

APPENDIX III : REMOTE ADAPTER STATUS

The remote adapter status processing consists of the following steps.

1. Obtain the address of the NDSE (remote machine) where the object (name) resides by executing resolve name function with NodeAdminTransport Address attribute set. This step is skipped if the address is already cached.
2. Send the adapter status request PDU, point-to-point, to the remote NDSE (only the fixed header).
3. The NDSE will process the adapter status request indication, and return the status information in the response PDU.

III.1 AdapterStatus Request PDU Format

The Adapter Status Request PDU will consist of a FIXED HEADER part as defined in the following table. It is the same fixed format as given in Figure 5.

Field	Value
Length Indicator	0000 0000
	0000 1010
Protocol Version	0001 0001
Type	0000 0010
Source Reference	variable
Flags	0000 0000
QOS	variable
Response Semantics	0000 0001
Response Code	0000 0000
Procedure Timeout	variable
Procedure	0000 1000

TABLE 23. AdapterStatus Request PDU Format

III.2 AdapterStatus Response PDU Format

The adapter status response PDU will consist of two parts, fixed part and variable part. The format for the fixed part will be the same as the request PDU but for the following changes:

Length will be length of the PDU following length indicator field.

Type will be set to response PDU, 0000 0100.

Response will be set to appropriate response code.

The variable part will consist of the following:

Response PDU Variable Part		
Field		
Length Indicator	2 Octets	
MAC Address	6 Octets	
External Option	1 Octet	
Result of Last Self Test	1 Octet	
Software Version	2 Octets	impl. specific
Reporting Period	2 Octets	
CRC Errors	2 Octets	
Alignment Errors	2 Octets	
Number of Collisions	2 Octets	
Number of Unsuccessful Xmit	2 Octets	
Frames Transmitted	4 Octets	
Frames Received	4 Octets	
Number of Retransmissions	2 Octets	
Resource Exhaustions	2 Octets	
Local Implementation	4 Octets	
Local Implementation	4 Octets	
Free NCBs	2 Octets	
Conf. Max NCBs	2 Octets	
Max NCBs	2 Octets	
Local Implementation	2 Octets	
Local Implementation	2 Octets	
Pending Sessions	2 Octets	
Conf. Max Sessions	2 Octets	
Max Sessions	2 Octets	
Max TPDU Size	2 Octets	
Quantity of Local Names	2 Octets	
List of Names	variable	
--Name	16 Octets	
--Name Number	1 Octet	
--Name Status	0xxx xxxx	unique name
	1xxx xxxx	group name
	xxxx x000	trying to register
	xxxx x100	registered
	xxxx x101	de-registered
	xxxx x110	duplicate name
	xxxx x111	duplicate name being de-registered

TABLE 24. Adapter Status Variable Part PDU Format

APPENDIX IV : WELL KNOWN ADDRESSES

There are several well known addresses, described here, for the effective operation of NetBIOS and the Name Service.

IV.1 Transport Selectors

1. NetBIOS Broadcast Address: T-Selector

   The NetBIOS broadcast address t-selector is defined as ''*'' followed by 15 blank spaces.

   T-Selector for Broadcast = ''*<15 spaces>''

2. All NetBIOS Directory Service Entities: T-Selector

   The NetBIOS Name Service Element for a Node, NetBIOS DIRECTORY SERVICE ENTITY (NDSE), will have a ''well known'' transport service access point identifier (t- selector), of 16 octet in length. This will be a reserved value.

   T-Selector for NDSE = ''*NetBIOS_NDSE<3 spaces>''

   Note that the choice of source NSAP address for the nodes that support multiple NSAPs is a local matter.

3. Recommended NDUA: T-Selector

   The recommended T-Selector, of 16 octets in length, for NDUAs on a network is given below. This will be a configurable parameter. The complete protocol address of the NDUA entities will be included in the ''I am Here PDUs''.

   Recommended T-Selector for NDUA = ''*NetBIOS_NDUA<3 spaces>''

IV.2 Network Layer Addresses

1. Group NSAP Address

   In order to implement group datagrams at transport level, only for intranet traffic, a special node number (station number) value is reserved in the network service access point address (NSAP Address). The same NSAP address will be used by the NDUA and NDSEs for their group datagrams.

   The group NSAP address will identify all the nodes on the given subnetwork.

   The general structure of the NSAP address, as per the TOP 3.0 specifications for binary syntax, is used here. Additional semantic constraints are described in the following two points.

   1. The recommended value of NSEL will be 01H, but it can be set to any other value as per the installation option.
   2. The station number field of NSAP address is set to group multicast address.

      STATION NUMBER [6 OCTETS] = 09006A000100H

   3. The format of the remaining DSP must be configurable following TOP 3.0 specifications.

   The NSAP address will use local AFI (49H) and the recommended format for full NSAP address will be[18]:

NSAP LENGTH [1 OCTET] = 10 (decimal)

NSAP = 49.nn.nn.09.00.6A.00.01.00.01

nn.nn=00.00, subnetwork number (default).

2. NSAP Address Formats

The general NSAP address formats will be as per the TOP 3.0 Specifications.

IV.3 Link Layer Addresses

1. Multicast/Functional Addresses

Multicast addresses (broadcast based LANs) will be used as the destination subnetwork point of attachment (SNPA) address for the group NSAP address defined in the previous item. The multicast address is given below. Note that the same functionality can also be achieved by using a broadcast address. Also, the recommended functional addresses used as multicast addresses in the token ring environment are provided. These are recommended values and must be configurable.

Function	Address
TOP/NetBIOS Multicast Address - IEEE 802.3	09.00.6A.00.01.00
End System Hellos (IS Address) - IEEE 802.5	C0.00.00.10.00.00
Intermediate System Hellos (ES Address) - IEEE 802.5	C0.00.00.08.00.00
TOP/NetBIOS Functional Address - IEEE 802.5	C0.00.00.20.00.00

TABLE 25. Recommended Multicast and Functional Addresses

2. LSAP Value

The LSAP value used by the NetBIOS Protocol for multicast and broadcast datagrams is given below. This is a recommended value and must be configurable.

ECH

18. Note that if the connected subnetwork is token ring, the multicast NSAP address maps to the functional address of C0.00.00.20 00.00 as the SNPA address.

APPENDIX V : OBJECTIVES AND ACTIONS OF NDUAs

The NetBIOS Directory User Agent provides two major support functions.

1. It helps in reducing the multicast traffic on the media.
2. It provides an interface to the OSI Directory Services for all the NetBIOS Entities.

V.1 New Protocol Element

One new protocol element is defined to support the NDUA functionalities, ''I am here PDU''.

I am here PDU is an advise PDU with:

- fixed header with ''I am here'' Procedure type and timeout=0, and
- protocol address (presentation address).

There is NO ''I am signing OFF'' PDU. The NDSEs can detect the absence of NDUAs based on timeouts. The actual implementation of this detection is a local matter.

V.2 Actions

There is a set of actions for NDUAs and NDSEs.

a. NDUAs multicast the ''I am here PDU'' N times at T intervals when joining the network.

b. NDSEs save the information received in ''I am here PDU'' of at least one NDUA.

c. NDSEs will multicast ''register name'' and ''resolve name'' requests if there is no known NDUA on the network, otherwise a point to point request is sent to one of the NDUAs.

d. NDUA will process the ''resolve name'' request by checking the cache, or making a request to the DUA and/or sending a multicast ''resolve name'' request if the entry is not available in cache.

   When processing a resolve name request for a group name, the NDUA may choose one among many transport addresses to put into its response. However, it is a local matter how NDUA makes this choice.

e. NDUAs will process the point-to-point ''register name'' request by checking in the local database (cache).

   - If the name is not found in the local database, the NDUA will multicast the ''register name'' request on the LOCAL network with the ResponseSemantics set to response on failure.
   - If it does not receive any response, that means the name is available and the registration request is granted and a success response is returned to the originator.
   - If the name was found in the local database and the presentation address supplied in the request is the same as that in the local database, the registration request is granted and a positive (success) response is returned to the originator.
   - If the name was found in the local database and the presentation address supplied in the request is different than in the local database, a point to point resolve name request is sent to the NDSE where the name was registered. If the resolve name succeeds, then the new register name request is denied, but if the resolve name request fails, then the register name request is granted. If the register name

request is granted then the local database is updated appropriately. Success or failure response is returned to the originator of the register name request. These checks are necessary as the NetBIOS objects can move from one machine to another, or nodes can go down and come back up without unregistering their services.

The NDUA will maintain in its database all the information (attributes) that was provided in the registration request PDU. Private attributes supplied with the register name request will be saved, and will be returned in the resolve name response PDU, if those attributes are requested. NDUA will not make any semantic analysis of this data.

f. Also, based on administrative ''filtering'', NDUAs will propagate the ''register name'' request to its DUA (if present). When such a request is propagated it will include the appropriate ''object-id'' attribute in the request to its DUA.

g. NDUAs will process the ''unregister name'' request by cleaning up the cache and propagating it to DUAs (point-to-point).

h. In addition, NDUAs will return point-to-point ''I am here PDU'' if an NDSE request is received as a multicast PDU. Note that this implies that an NDUA must receive datagrams that were sent to NDSE's t- selector (*NetBIOS_NDSE).

i. NDSEs will multicast the ''resolve name'' request (N- X) times if no responses are received for the first ''X'' point-to-point requests.

j. NDUAs will set the NDUA FLAG in all the PDUs generated by them. Thus other NDUAs, if present on the subnetwork, can distinguish between the PDUs received from NDUAs and from NDSEs. The NDSEs will always reset the NDUA flag in all PDUs generated by them, and they do not attach any semantic meaning to this flag in the received PDUs.

V.3 Object Class Definition

The following NetBIOS Object Class is defined for use in conjunction with OSI Directory Services.

```
NetBIOSEntity OBJECT-CLASS
    SUBCLASS OF top
    MUST CONTAIN {
        NetBIOSName
        nameType
        presentationAddressSet
        adminPresentationAddress
    }
::= {NetBIOSEntity-object-identifier-value}

NetBIOSName ATTRIBUTE
    WITH ATTRIBUTE SYNTAX
        octetStringSyntax (SIZE(16))
::= {netBIOSName-object-identifier-value}

nameType ATTRIBUTE
    WITH ATTRIBUTE SYNTAX
        INTEGER{
            group (0),
            unique (1)
        }
        MATCHES FOR EQUALITY
        SINGLE VALUE
::= {nameType-object-identifier-value}

presentationAddressSet ::=
    SET OF PresentationAddress

adminPresentationAddress ::=
    PresentationAddress
```

Note that the above definition, object identifier value, must be assigned by an OSI Registration Authority. Currently the U.S. does not have one, though it will likely be NIST (previously known as NBS) or ANSI. Once a registration authority is set up, it will be requested to assign the value.

APPENDIX VI : ERRATA AND CLARIFICATION

For future use.

Appendix F

RFC 1001

This appendix reproduces, in full and unedited, RFC 1001, Protocol Standard for a NetBIOS Service on a TCP/UDP Transport: Concepts and Methods.

Network Working Group
Request for Comments: 1001 March, 1987

PROTOCOL STANDARD FOR A NetBIOS SERVICE ON A TCP/UDP TRANSPORT: CONCEPTS AND METHODS

ABSTRACT

This RFC defines a proposed standard protocol to support NetBIOS services in a TCP/IP environment. Both local network and internet operation are supported. Various node types are defined to accommodate local and internet topologies and to allow operation with or without the use of IP broadcast.

This RFC describes the NetBIOS-over-TCP protocols in a general manner, emphasizing the underlying ideas and techniques. Detailed specifications are found in a companion RFC, "Protocol Standard For a NetBIOS Service on a TCP/UDP Transport: Detailed Specifications".

SUMMARY OF CONTENTS

TABLE OF CONTENTS

PROTOCOL STANDARD FOR A NetBIOS SERVICE ON A TCP/UDP TRANSPORT: CONCEPTS AND METHODS

1. STATUS OF THIS MEMO

This RFC specifies a proposed standard for the Internet community. Since this topic is new to the Internet community, discussions and suggestions are specifically requested.

Please send written comments to:

Karl Auerbach
Epilogue Technology Corporation
P.O. Box 5432
Redwood City, CA 94063

Please send online comments to:

Avnish Aggarwal
Internet: mtxinu!excelan!avnish@ucbvax.berkeley.edu
Usenet: ucbvax!mtxinu!excelan!avnish

Distribution of this document is unlimited.

2. ACKNOWLEDGEMENTS

This RFC has been developed under the auspices of the Internet Activities Board, especially the End-to-End Services Task Force.

The following individuals have contributed to the development of this RFC:

Avnish Aggarwal	Arvind Agrawal	Lorenzo Aguilar
Geoffrey Arnold	Karl Auerbach	K. Ramesh Babu
Keith Ball	Amatzia Ben-Artzi	Vint Cerf
Richard Cherry	David Crocker	Steve Deering
Greg Ennis	Steve Holmgren	Jay Israel
David Kaufman	Lee LaBarre	James Lau
Dan Lynch	Gaylord Miyata	David Stevens
Steve Thomas	Ishan Wu	

The system proposed by this RFC does not reflect any existing Netbios-over-TCP implementation. However, the design incorporates considerable knowledge obtained from prior implementations. Special thanks goes to the following organizations which have provided this invaluable information:

CMC/Syros Excelan Sytek Ungermann-Bass

3. INTRODUCTION

This RFC describes the ideas and general methods used to provide NetBIOS on a TCP and UDP foundation. A companion RFC, "Protocol Standard For a NetBIOS Service on a TCP/UDP Transport: Detailed Specifications"[1] contains detailed descriptions of packet formats, protocols, and defined constants and variables.

The NetBIOS service has become the dominant mechanism for personal computer networking. NetBIOS provides a vendor independent interface for the IBM Personal Computer (PC) and compatible systems.

NetBIOS defines a software interface not a protocol. There is no "official" NetBIOS service standard. In practice, however, the IBM PC-Network version is used as a reference. That version is described in the IBM document 6322916, "Technical Reference PC Network"[2].

Protocols supporting NetBIOS services have been constructed on diverse protocol and hardware foundations. Even when the same foundation is used, different implementations may not be able to interoperate unless they use a common protocol. To allow NetBIOS interoperation in the Internet, this RFC defines a standard protocol to support NetBIOS services using TCP and UDP.

NetBIOS has generally been confined to personal computers to date. However, since larger computers are often well suited to run certain NetBIOS applications, such as file servers, this specification has been designed to allow an implementation to be built on virtually any type of system where the TCP/IP protocol suite is available.

This standard defines a set of protocols to support NetBIOS services.

These protocols are more than a simple communications service involving two entities. Rather, this note describes a distributed system in which many entities play a part even if they are not involved as an end-point of a particular NetBIOS connection.

This standard neither constrains nor determines how those services are presented to application programs.

Nevertheless, it is expected that on computers operating under the PC-DOS and MS-DOS operating systems that the existing NetBIOS interface will be preserved by implementors.

NOTE: Various symbolic values are used in this document. For their definitions, refer to the Detailed Specifications[1].

4. DESIGN PRINCIPLES

In order to develop the specification the following design principles were adopted to guide the effort. Most are typical to any protocol standardization effort; however, some have been assigned priorities that may be considered unusual.

4.1. PRESERVE NetBIOS SERVICES

In the absence of an "official" standard for NetBIOS services, the version found in the IBM PC Network Technical Reference[2] is used.

NetBIOS is the foundation of a large body of existing applications. It is desirable to operate these applications on TCP networks and to extend them beyond personal computers into larger hosts. To support these applications, NetBIOS on TCP must closely conform to the services offered by existing NetBIOS systems.

IBM PC-Network NetBIOS contains some implementation specific characteristics. This standard does not attempt to completely preserve these. It is certain that some existing software requires these characteristics and will fail to operate correctly on a NetBIOS service based on this RFC.

4.2. USE EXISTING STANDARDS

Protocol development, especially with standardization, is a demanding process. The development of new protocols must be minimized.

It is considered essential that an existing standard which provides the necessary functionality with reasonable performance always be chosen in preference to developing a new protocol.

When a standard protocol is used, it must be unmodified.

4.3. MINIMIZE OPTIONS

The standard for NetBIOS on TCP should contain few, if any, options.

Where options are included, the options should be designed so that devices with different option selections should interoperate.

4.4. TOLERATE ERRORS AND DISRUPTIONS

NetBIOS networks typically operate in an uncontrolled environment. Computers come on-line at arbitrary times. Computers usually go off-line without any notice to their peers. The software is often operated by users who are unfamiliar with networks and who may randomly perturb configuration settings.

Despite this chaos, NetBIOS networks work. NetBIOS on TCP must also

be able to operate well in this environment.

Robust operation does not necessarily mean that the network is proof against all disruptions. A typical NetBIOS network may be disrupted by certain types of behavior, whether inadvertent or malicious.

4.5. DO NOT REQUIRE CENTRAL MANAGEMENT

NetBIOS on TCP should be able to operate, if desired, without centralized management beyond that typically required by a TCP based network.

4.6. ALLOW INTERNET OPERATION

The proposed standard recognizes the need for NetBIOS operation across a set of networks interconnected by network (IP) level relays (gateways.)

However, the standard assumes that this form of operation will be less frequent than on the local MAC bridged-LAN.

4.7. MINIMIZE BROADCAST ACTIVITY

The standard pre-supposes that the only broadcast services are those supported by UDP. Multicast capabilities are not assumed to be available in any form.

Despite the availability of broadcast capabilities, the standard recognizes that some administrations may wish to avoid heavy broadcast activity. For example, an administration may wish to avoid isolated non-participating hosts from the burden of receiving and discarding NetBIOS broadcasts.

4.8. PERMIT IMPLEMENTATION ON EXISTING SYSTEMS

The NetBIOS on TCP protocol should be implementable on common operating systems, such as Unix(tm) and VAX/VMS(tm), without massive effort.

The NetBIOS protocols should not require services typically unavailable on presently existing TCP/UDP/IP implementations.

4.9. REQUIRE ONLY THE MINIMUM NECESSARY TO OPERATE

The protocol definition should specify only the minimal set of protocols required for interoperation. However, additional protocol elements may be defined to enhance efficiency. These latter elements may be generated at the option of the sender, although they must be accepted by all receivers.

4.10. MAXIMIZE EFFICIENCY

To be useful, a protocol must conduct its business quickly.

4.11. MINIMIZE NEW INVENTIONS

When an existing protocol is not quite able to support a necessary function, but with a small amount of change, it could, that protocol should be used. This is felt to be easier to achieve than development of new protocols; further, it is likely to have more general utility for the Internet.

5. OVERVIEW OF NetBIOS

This section describes the NetBIOS services. It is for background information only. The reader may chose to skip to the next section.

NetBIOS was designed for use by groups of PCs, sharing a broadcast medium. Both connection (Session) and connectionless (Datagram) services are provided, and broadcast and multicast are supported. Participants are identified by name. Assignment of names is distributed and highly dynamic.

NetBIOS applications employ NetBIOS mechanisms to locate resources, establish connections, send and receive data with an application peer, and terminate connections. For purposes of discussion, these mechanisms will collectively be called the NetBIOS Service.

This service can be implemented in many different ways. One of the first implementations was for personal computers running the PC-DOS and MS-DOS operating systems. It is possible to implement NetBIOS within other operating systems, or as processes which are, themselves, simply application programs as far as the host operating system is concerned.

The NetBIOS specification, published by IBM as "Technical Reference PC Network"[2] defines the interface and services available to the NetBIOS user. The protocols outlined by that document pertain only to the IBM PC Network and are not generally applicable to other networks.

5.1. INTERFACE TO APPLICATION PROGRAMS

NetBIOS on personal computers includes both a set of services and an exact program interface to those services. NetBIOS on other computer systems may present the NetBIOS services to programs using other interfaces. Except on personal computers, no clear standard for a NetBIOS software interface has emerged.

5.2. NAME SERVICE

NetBIOS resources are referenced by name. Lower-level address information is not available to NetBIOS applications. An application, representing a resource, registers one or more names that it wishes to use.

The name space is flat and uses sixteen alphanumeric characters. Names may not start with an asterisk (*).

Registration is a bid for use of a name. The bid may be for exclusive (unique) or shared (group) ownership. Each application contends with the other applications in real time. Implicit permission is granted to a station when it receives no objections. That is, a bid is made and the application waits for a period of time. If no objections are received, the station assumes that it has permission.

A unique name should be held by only one station at a time. However, duplicates ("name conflicts") may arise due to errors.

All instances of a group name are equivalent.

An application referencing a name generally does not know (or care) whether the name is registered as a unique or a group name.

An explicit name deletion function is specified, so that applications may remove a name. Implicit name deletion occurs when a station ceases operation. In the case of personal computers, implicit name deletion is a frequent occurrence.

The Name Service primitives are:

1) Add Name

   The requesting application wants exclusive use of the name.

2) Add Group Name

   The requesting application is willing to share use of the name with other applications.

3) Delete Name

   The application no longer requires use of the name. It is important to note that typical use of NetBIOS is among independently-operated personal computers. A common way to stop using a PC is to turn it off; in this case, the graceful give-back mechanism, provided by the Delete Name function, is not used. Because this occurs frequently, the network service must support this behavior.

5.3. SESSION SERVICE

A session is a reliable message exchange, conducted between a pair of NetBIOS applications. Sessions are full-duplex, sequenced, and reliable. Data is organized into messages. Each message may range in size from 0 to 131,071 bytes. No expedited or urgent data capabilities are present.

Multiple sessions may exist between any pair of calling and called names.

The parties to a connection have access to the calling and called names.

The NetBIOS specification does not define how a connection request to a shared (group) name resolves into a session. The usual assumption is that a session may be established with any one owner of the called group name.

An important service provided to NetBIOS applications is the detection of sessions failure. The loss of a session is reported to an application via all of the outstanding service requests for that session. For example, if the application has only a NetBIOS receive primitive pending and the session terminates, the pending receive will abort with a termination indication.

Session Service primitives are:

1) Call

   Initiate a session with a process that is listening under the specified name. The calling entity must indicate both a calling name (properly registered to the caller) and a called name.

2) Listen

   Accept a session from a caller. The listen primitive may be constrained to accept an incoming call from a named caller. Alternatively, a call may be accepted from any caller.

3) Hang Up

   Gracefully terminate a session. All pending data is transferred before the session is terminated.

4) Send

   Transmit one message. A time-out can occur. A time-out of any session send forces the non-graceful termination of the session.

A "chain send" primitive is required by the PC NetBIOS software interface to allow a single message to be gathered from pieces in various buffers. Chain Send is an interface detail and does not effect the protocol.

5) Receive

Receive data. A time-out can occur. A time-out on a session receive only terminates the receive, not the session, although the data is lost.

The receive primitive may be implemented with variants, such as "Receive Any", which is required by the PC NetBIOS software interface. Receive Any is an interface detail and does not effect the protocol.

6) Session Status

Obtain information about all of the requestor's sessions, under the specified name. No network activity is involved.

5.4. DATAGRAM SERVICE

The Datagram service is an unreliable, non-sequenced, connectionless service. Datagrams are sent under cover of a name properly registered to the sender.

Datagrams may be sent to a specific name or may be explicitly broadcast.

Datagrams sent to an exclusive name are received, if at all, by the holder of that name. Datagrams sent to a group name are multicast to all holders of that name. The sending application program cannot distinguish between group and unique names and thus must act as if all non-broadcast datagrams are multicast.

As with the Session Service, the receiver of the datagram is told the sending and receiving names.

Datagram Service primitives are:

1) Send Datagram

Send an unreliable datagram to an application that is associated with the specified name. The name may be unique or group; the sender is not aware of the difference. If the name belongs to a group, then each member is to receive the datagram.

2) Send Broadcast Datagram

Send an unreliable datagram to any application with a Receive Broadcast Datagram posted.

3) Receive Datagram

Receive a datagram sent by a specified originating name to the specified name. If the originating name is an asterisk, then the datagram may have been originated under any name.

Note: An arriving datagram will be delivered to all pending Receiving Datagrams that have source and destination specifications matching those of the datagram. In other words, if a program (or group of programs) issue a series of identical Receive Datagrams, one datagram will cause the entire series to complete.

4) Receive Broadcast Datagram

Receive a datagram sent as a broadcast.

If there are multiple pending Receive Broadcast Datagram operations pending, all will be satisfied by the same received datagram.

5.5. MISCELLANEOUS FUNCTIONS

The following functions are present to control the operation of the hardware interface to the network. These functions are generally implementation dependent.

1) Reset

Initialize the local network adapter.

2) Cancel

Abort a pending NetBIOS request. The successful cancel of a Send (or Chain Send) operation will terminate the associated session.

3) Adapter Status

Obtain information about the local network adapter or of a remote adapter.

4) Unlink

For use with Remote Program Load (RPL). Unlink redirects the PC boot disk device back to the local disk. See the

NetBIOS specification for further details concerning RPL and the Unlink operation (see page 2-35 in [2]).

5) Remote Program Load

Remote Program Load (RPL) is not a NetBIOS function. It is a NetBIOS application defined by IBM in their NetBIOS specification (see pages 2-80 through 2-82 in [2]).

5.6. NON-STANDARD EXTENSIONS

The IBM Token Ring implementation of NetBIOS has added at least one new user capability:

1) Find Name

This function determines whether a given name has been registered on the network.

6. NetBIOS FACILITIES SUPPORTED BY THIS STANDARD

The protocol specified by this standard permits an implementer to provide all of the NetBIOS services as described in the IBM "Technical Reference PC Network"[2].

The following NetBIOS facilities are outside the scope of this specification. These are local implementation matters and do not impact interoperability:

- RESET
- SESSION STATUS
- UNLINK
- RPL (Remote Program Load)

7. REQUIRED SUPPORTING SERVICE INTERFACES AND DEFINITIONS

The protocols described in this RFC require service interfaces to the following:

- TCP[3,4]
- UDP[5]

Byte ordering, addressing conventions (including addresses to be used for broadcasts and multicasts) are defined by the most recent version of:

- Assigned Numbers[6]

Additional definitions and constraints are in:

- IP[7]
- Internet Subnets[8,9,10]

8. RELATED PROTOCOLS AND SERVICES

The design of the protocols described in this RFC allow for the future incorporation of the following protocols and services. However, before this may occur, certain extensions may be required to the protocols defined in this RFC or to those listed below.

- Domain Name Service[11,12,13,14]
- Internet Group Multicast[15,16]

9. NetBIOS SCOPE

A "NetBIOS Scope" is the population of computers across which a registered NetBIOS name is known. NetBIOS broadcast and multicast datagram operations must reach the entire extent of the NetBIOS scope.

An internet may support multiple, non-intersecting NetBIOS Scopes.

Each NetBIOS scope has a "scope identifier". This identifier is a character string meeting the requirements of the domain name system for domain names.

NOTE: Each implementation of NetBIOS-over-TCP must provide mechanisms to manage the scope identifier(s) to be used.

Control of scope identifiers implies a requirement for additional NetBIOS interface capabilities. These may be provided through extensions of the user service interface or other means (such as node configuration parameters.) The nature of these extensions is not part of this specification.

10. NetBIOS END-NODES

End-nodes support NetBIOS service interfaces and contain applications.

Three types of end-nodes are part of this standard:

- Broadcast ("B") nodes
- Point-to-point ("P") nodes
- Mixed mode ("M") nodes

An IP address may be associated with only one instance of one of the above types.

Without having preloaded name-to-address tables, NetBIOS participants

are faced with the task of dynamically resolving references to one another. This can be accomplished with broadcast or mediated point-to-point communications.

B nodes use local network broadcasting to effect a rendezvous with one or more recipients. P and M nodes use the NetBIOS Name Server (NBNS) and the NetBIOS Datagram Distribution Server (NBDD) for this same purpose.

End-nodes may be combined in various topologies. No matter how combined, the operation of the B, P, and M nodes is not altered.

NOTE: It is recommended that the administration of a NetBIOS scope avoid using both M and B nodes within the same scope. A NetBIOS scope should contain only B nodes or only P and M nodes.

10.1. BROADCAST (B) NODES

Broadcast (or "B") nodes communicate using a mix of UDP datagrams (both broadcast and directed) and TCP connections. B nodes may freely interoperate with one another within a broadcast area. A broadcast area is a single MAC-bridged "B-LAN". (See Appendix A for a discussion of using Internet Group Multicasting as a means to extend a broadcast area beyond a single B-LAN.)

10.2. POINT-TO-POINT (P) NODES

Point-to-point (or "P") nodes communicate using only directed UDP datagrams and TCP sessions. P nodes neither generate nor listen for broadcast UDP packets. P nodes do, however, offer NetBIOS level broadcast and multicast services using capabilities provided by the NBNS and NBDD.

P nodes rely on NetBIOS name and datagram distribution servers. These servers may be local or remote; P nodes operate the same in either case.

10.3. MIXED MODE (M) NODES

Mixed mode nodes (or "M") nodes are P nodes which have been given certain B node characteristics. M nodes use both broadcast and unicast. Broadcast is used to improve response time using the assumption that most resources reside on the local broadcast medium rather than somewhere in an internet.

M nodes rely upon NBNS and NBDD servers. However, M nodes may continue limited operation should these servers be temporarily unavailable.

11. NetBIOS SUPPORT SERVERS

Two types of support servers are part of this standard:

- NetBIOS name server ("NBNS") nodes
- Netbios datagram distribution ("NBDD") nodes

NBNS and NBDD nodes are invisible to NetBIOS applications and are part of the underlying NetBIOS mechanism.

NetBIOS name and datagram distribution servers are the focus of name and datagram activity for P and M nodes.

Both the name (NBNS) and datagram distribution (NBDD) servers are permitted to shift part of their operation to the P or M end-node which is requesting a service.

Since the assignment of responsibility is dynamic, and since P and M nodes must be prepared to operate should the NetBIOS server delegate control to the maximum extent, the system naturally accommodates improvements in NetBIOS server function. For example, as Internet Group Multicasting becomes more widespread, new NBDD implementations may elect to assume full responsibility for NetBIOS datagram distribution.

Interoperability between different implementations is assured by imposing requirements on end-node implementations that they be able to accept the full range of legal responses from the NBNS or NBDD.

11.1. NetBIOS NAME SERVER (NBNS) NODES

The NBNS is designed to allow considerable flexibility with its degree of responsibility for the accuracy and management of NetBIOS names. On one hand, the NBNS may elect not to accept full responsibility, leaving the NBNS essentially a "bulletin board" on which name/address information is freely posted (and removed) by P and M nodes without validation by the NBNS. Alternatively, the NBNS may elect to completely manage and validate names. The degree of responsibility that the NBNS assumes is asserted by the NBNS each time a name is claimed through a simple mechanism. Should the NBNS not assert full control, the NBNS returns enough information to the requesting node so that the node may challenge any putative holder of the name.

This ability to shift responsibility for NetBIOS name management between the NBNS and the P and M nodes allows a network administrator (or vendor) to make a tradeoff between NBNS simplicity, security, and delay characteristics.

A single NBNS may be implemented as a distributed entity, such as the Domain Name Service. However, this RFC does not attempt to define

the internal communications which would be used.

11.1.1. RELATIONSHIP OF THE NBNS TO THE DOMAIN NAME SYSTEM

The NBNS design attempts to align itself with the Domain Name System in a number of ways.

First, the NetBIOS names are encoded in a form acceptable to the domain name system.

Second, a scope identifier is appended to each NetBIOS name. This identifier meets the restricted character set of the domain system and has a leading period. This makes the NetBIOS name, in conjunction with its scope identifier, a valid domain system name.

Third, the negotiated responsibility mechanisms permit the NBNS to be used as a simple bulletin board on which are posted (name,address) pairs. This parallels the existing domain sytem query service.

This RFC, however, requires the NBNS to provide services beyond those provided by the current domain name system. An attempt has been made to coalesce all the additional services which are required into a set of transactions which follow domain name system styles of interaction and packet formats.

Among the areas in which the domain name service must be extended before it may be used as an NBNS are:

- Dynamic addition of entries
- Dynamic update of entry data
- Support for multiple instance (group) entries
- Support for entry time-to-live values and ability to accept refresh messages to restart the time-to-live period
- New entry attributes

11.2. NetBIOS DATAGRAM DISTRIBUTION SERVER (NBDD) NODES

The internet does not yet support broadcasting or multicasting. The NBDD extends NetBIOS datagram distribution service to this environment.

The NBDD may elect to complete, partially complete, or totally refuse to service a node's request to distribute a NetBIOS datagram. An end-node may query an NBDD to determine whether the NBDD will deliver a datagram to a specific NetBIOS name.

The design of NetBIOS-over-TCP lends itself to the use of Internet Group Multicast. For details see Appendix A.

11.3. RELATIONSHIP OF NBNS AND NBDD NODES

This RFC defines the NBNS and NBDD as distinct, separate entities.

In the absence of NetBIOS name information, a NetBIOS datagram distribution server must send a copy to each end-node within a NetBIOS scope.

An implementer may elect to construct NBNS and NBDD nodes which have a private protocol for the exchange of NetBIOS name information. Alternatively, an NBNS and NBDD may be implemented within the same device.

NOTE: Implementations containing private NBNS-NBDD protocols or combined NBNS-NBDD functions must be clearly identified.

11.4. RELATIONSHIP OF NetBIOS SUPPORT SERVERS AND B NODES

As defined in this RFC, neither NBNS nor NBDD nodes interact with B nodes. NetBIOS servers do not listen to broadcast traffic on any broadcast area to which they may be attached. Nor are the NetBIOS support servers even aware of B node activities or names claimed or used by B nodes.

It may be possible to extend both the NBNS and NBDD so that they participate in B node activities and act as a bridge to P and M nodes. However, such extensions are beyond the scope of this specification.

12. TOPOLOGIES

B, P, M, NBNS, and NBDD nodes may be combined in various ways to form useful NetBIOS environments. This section describes some of these combinations.

There are three classes of operation:

- Class 0: B nodes only.
- Class 1: P nodes only.
- Class 2: P and M nodes together.

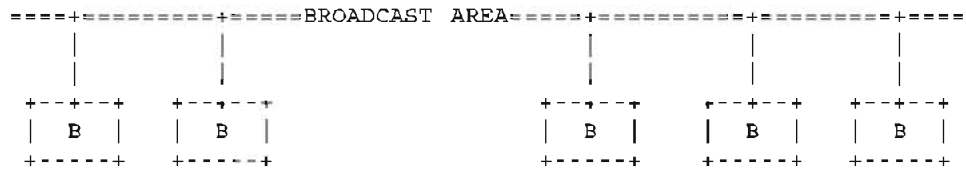
In the drawings which follow, any P node may be replaced by an M node. The effects of such replacement will be mentioned in conjunction with each example below.

12.1. LOCAL

A NetBIOS scope is operating locally when all entities are within the same broadcast area.

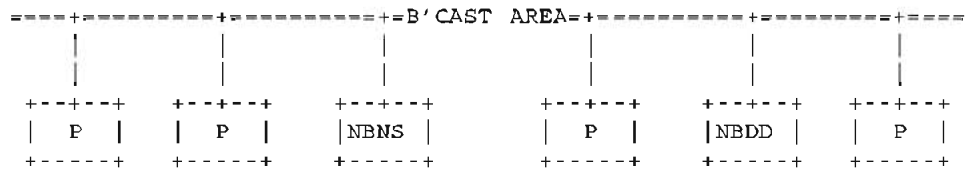
12.1.1. B NODES ONLY

Local operation with only B nodes is the most basic mode of operation. Name registration and discovery procedures use broadcast mechanisms. The NetBIOS scope is limited by the extent of the broadcast area. This configuration does not require NetBIOS support servers.



12.1.2. P NODES ONLY

This configuration would typically be used when the network administrator desires to eliminate NetBIOS as a source of broadcast activity.



This configuration operates the same as if it were in an internet and is cited here only due to its convenience as a means to reduce the use of broadcast.

Replacement of one or more of the P nodes with M nodes will not affect the operation of the other P and M nodes. P and M nodes will be able to interact with one another. Because M nodes use broadcast, overall broadcast activity will increase.

12.1.3. MIXED B AND P NODES

B and P nodes do not interact with one another. Replacement of P nodes with M nodes will allow B's and M's to interact.

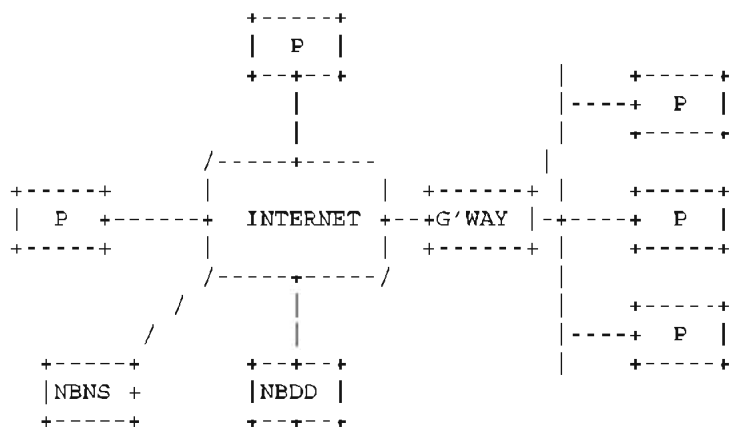
NOTE: B nodes and M nodes may be intermixed only on a local broadcast area. B and M nodes should not be intermixed in an internet environment.

12.2. INTERNET

12.2.1. P NODES ONLY

P nodes may be scattered at various locations in an internetwork. They require both an NBNS and an NBDD for NetBIOS name and datagram support, respectively.

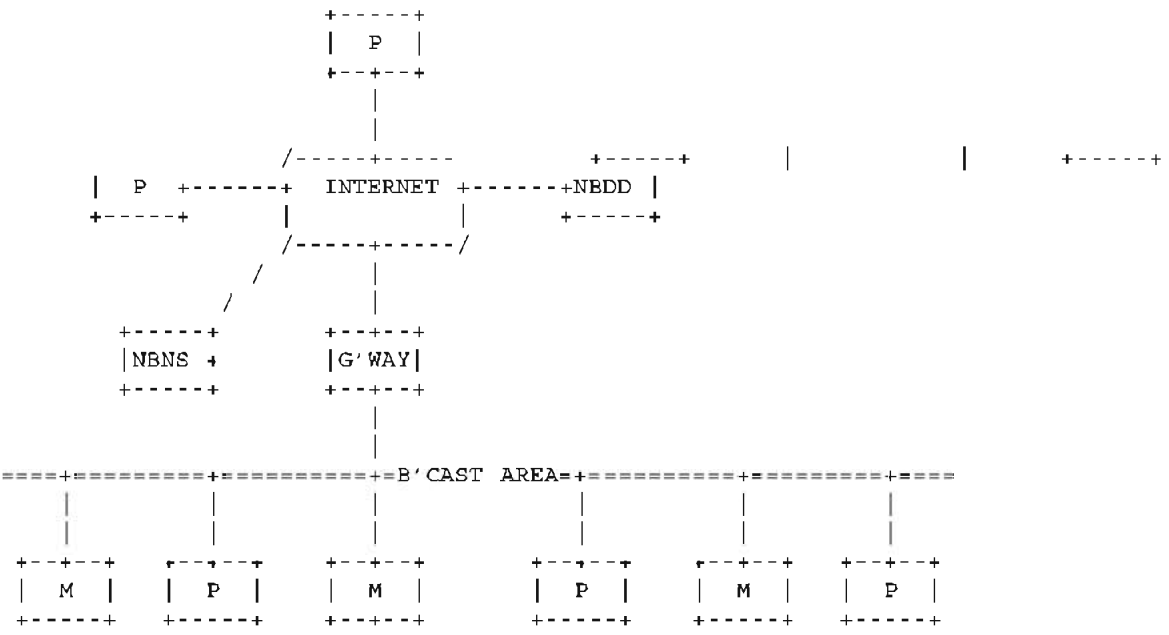
The NetBIOS scope is determined by the NetBIOS scope identifier (domain name) used by the various P (and M) nodes. An internet may contain numerous NetBIOS scopes.



Any P node may be replaced by an M node with no loss of function to any node. However, broadcast activity will be increased in the broadcast area to which the M node is attached.

12.2.2. MIXED M AND P NODES

M and P nodes may be mixed. When locating NetBIOS names, M nodes will tend to find names held by other M nodes on the same common broadcast area in preference to names held by P nodes or M nodes elsewhere in the network.



NOTE: B and M nodes should not be intermixed in an internet environment. Doing so would allow undetected NetBIOS name conflicts to arise and cause unpredictable behavior.

13. GENERAL METHODS

Overlying the specific protocols, described later, are a few general methods of interaction between entities.

13.1. REQUEST/RESPONSE INTERACTION STYLE

Most interactions between entities consist of a request flowing in one direction and a subsequent response flowing in the opposite direction.

In those situations where interactions occur on unreliable transports (i.e. UDP) or when a request is broadcast, there may not be a strict interlocking or one-to-one relationship between requests and responses.

In no case, however, is more than one response generated for a received request. While a response is pending the responding entity may send one or more wait acknowledgements.

13.1.1. RETRANSMISSION OF REQUESTS

UDP is an unreliable delivery mechanism where packets can be lost, received out of transmit sequence, duplicated and delivery can be significantly delayed. Since the NetBIOS protocols make heavy use of UDP, they have compensated for its unreliability with extra mechanisms.

Each NetBIOS packet contains all the necessary information to process it. None of the protocols use multiple UDP packets to convey a single request or response. If more information is required than will fit in a single UDP packet, for example, when a P-type node wants all the owners of a group name from a NetBIOS server, a TCP connection is used. Consequently, the NetBIOS protocols will not fail because of out of sequence delivery of UDP packets.

To overcome the loss of a request or response packet, each request operation will retransmit the request if a response is not received within a specified time limit.

Protocol operations sensitive to successive response packets, such as name conflict detection, are protected from duplicated packets because they ignore successive packets with the same NetBIOS information. Since no state on the responder's node is associated with a request, the responder just sends the appropriate response whenever a request packet arrives. Consequently, duplicate or delayed request packets have no impact.

For all requests, if a response packet is delayed too long another request packet will be transmitted. A second response packet being sent in response to the second request packet is equivalent to a duplicate packet. Therefore, the protocols will ignore the second packet received. If the delivery of a response is delayed until after the request operation has been completed, successfully or not, the response packet is ignored.

13.1.2. REQUESTS WITHOUT RESPONSES: DEMANDS

Some request types do not have matching responses. These requests are known as "demands". In general a "demand" is an imperative request; the receiving node is expected to obey. However, because demands are unconfirmed, they are used only in situations where, at most, limited damage would occur if the demand packet should be lost.

Demand packets are not retransmitted.

13.2. TRANSACTIONS

Interactions between a pair of entities are grouped into "transactions". These transactions comprise one or more request/response pairs.

13.2.1. TRANSACTION ID

Since multiple simultaneous transactions may be in progress between a pair of entities a "transaction id" is used.

The originator of a transaction selects an ID unique to the originator. The transaction id is reflected back and forth in each interaction within the transaction. The transaction partners must match responses and requests by comparison of the transaction ID and the IP address of the transaction partner. If no matching request can be found the response must be discarded.

A new transaction ID should be used for each transaction. A simple 16 bit transaction counter ought to be an adequate id generator. It is probably not necessary to search the space of outstanding transaction ID to filter duplicates: it is extremely unlikely that any transaction will have a lifetime that is more than a small fraction of the typical counter cycle period. Use of the IP addresses in conjunction with the transaction ID further reduces the possibility of damage should transaction IDs be prematurely re-used.

13.3. TCP AND UDP FOUNDATIONS

This version of the NetBIOS-over-TCP protocols uses UDP for many interactions. In the future this RFC may be extended to permit such interactions to occur over TCP connections (perhaps to increase efficiency when multiple interactions occur within a short time or when NetBIOS datagram traffic reveals that an application is using NetBIOS datagrams to support connection- oriented service.)

14. REPRESENTATION OF NETBIOS NAMES

NetBIOS names as seen across the client interface to NetBIOS are exactly 16 bytes long. Within the NetBIOS-over-TCP protocols, a longer representation is used.

There are two levels of encoding. The first level maps a NetBIOS name into a domain system name. The second level maps the domain system name into the "compressed" representation required for interaction with the domain name system.

Except in one packet, the second level representation is the only NetBIOS name representation used in NetBIOS-over-TCP packet formats. The exception is the RDATA field of a NODE STATUS RESPONSE packet.

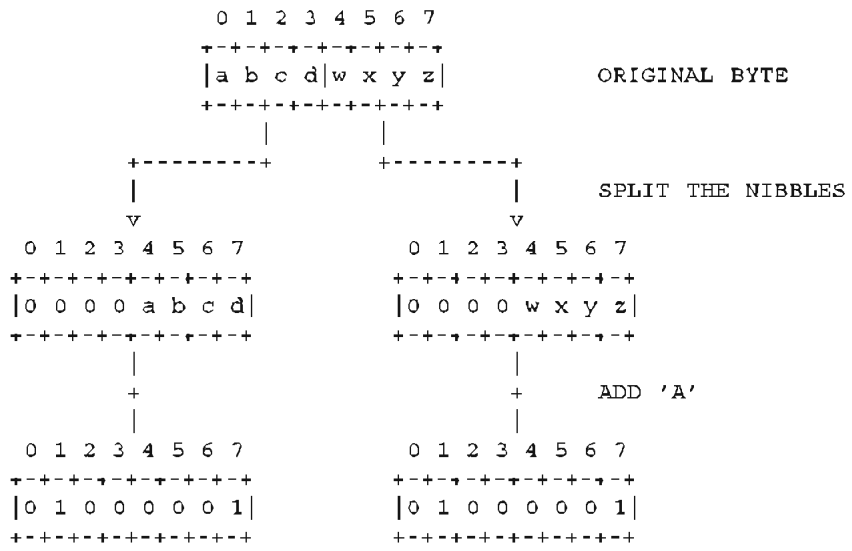
14.1. FIRST LEVEL ENCODING

The first level representation consists of two parts:

- NetBIOS name
- NetBIOS scope identifier

The 16 byte NetBIOS name is mapped into a 32 byte wide field using a reversible, half-ASCII, biased encoding. Each half-octet of the NetBIOS name is encoded into one byte of the 32 byte field. The first half octet is encoded into the first byte, the second half-octet into the second byte, etc.

Each 4-bit, half-octet of the NetBIOS name is treated as an 8-bit, right-adjusted, zero-filled binary number. This number is added to value of the ASCII character 'A' (hexidecimal 41). The resulting 8-bit number is stored in the appropriate byte. The following diagram demonstrates this procedure:



This encoding results in a NetBIOS name being represented as a sequence of 32 ASCII, upper-case characters from the set {A,B,C...N,O,P}.

The NetBIOS scope identifier is a valid domain name (without a leading dot).

An ASCII dot (2E hexidecimal) and the scope identifier are appended to the encoded form of the NetBIOS name, the result forming a valid domain name.

For example, the NetBIOS name "The NetBIOS name" in the NetBIOS scope "SCOPE.ID.COM" would be represented at level one by the ASCII character string:

FEGHGFCAEOGFHEECEJEPFDCAHEGBGNGF.SCOPE.ID.COM

14.2. SECOND LEVEL ENCODING

The first level encoding must be reduced to second level encoding. This is performed according to the rules defined in on page 31 of RFC 883[12] in the section on "Domain name representation and compression". Also see the section titled "Name Formats" in the Detailed Specifications[1].

15. NetBIOS NAME SERVICE

Before a name may be used, the name must be registered by a node. Once acquired, the name must be defended against inconsistent registration by other nodes. Before building a NetBIOS session or sending a NetBIOS datagram, the one or more holders of the name must be located.

The NetBIOS name service is the collection of procedures through which nodes acquire, defend, and locate the holders of NetBIOS names.

The name service procedures are different depending whether the end-node is of type B, P, or M.

15.1. OVERVIEW OF NetBIOS NAME SERVICE

15.1.1. NAME REGISTRATION (CLAIM)

Each NetBIOS node can own more than one name. Names are acquired dynamically through the registration (name claim) procedures.

Every node has a permanent unique name. This name, like any other name, must be explicitly registered by all end-node types.

A name can be unique (exclusive) or group (non-exclusive). A unique name may be owned by a single node; a group name may be owned by any number of nodes. A name ceases to exist when it is not owned by at least one node. There is no intrinsic quality of a name which determines its characteristics: these are established at the time of registration.

Each node maintains state information for each name it has registered. This information includes:

- Whether the name is a group or unique name
- Whether the name is "in conflict"
- Whether the name is in the process of being deleted

B nodes perform name registration by broadcasting claim requests, soliciting a defense from any node already holding the name.

P nodes perform name registration through the agency of the NBNS.

M nodes register names through an initial broadcast, like B nodes, then, in the absence of an objection, by following the same procedures as a P node. In other words, the broadcast action may terminate the attempt, but is not sufficient to confirm the registration.

15.1.2. NAME QUERY (DISCOVERY)

Name query (also known as "resolution" or "discovery") is the procedure by which the IP address(es) associated with a NetBIOS name are discovered. Name query is required during the following operations:

During session establishment, calling and called names must be specified. The calling name must exist on the node that posts the CALL. The called name must exist on a node that has previously posted a LISTEN. Either name may be a unique or group name.

When a directed datagram is sent, a source and destination name must be specified. If the destination name is a group name, a datagram is sent to all the members of that group.

Different end-node types perform name resolution using different techniques, but using the same packet formats:

- B nodes solicit name information by broadcasting a request.
- P nodes ask the NBNS.
- M nodes broadcast a request. If that does not provide the desired information, an inquiry is sent to the NBNS.

15.1.3. NAME RELEASE

NetBIOS names may be released explicitly or silently by an end- node. Silent release typically occurs when an end-node fails or is turned-off. Most of the mechanisms described below are present to detect silent name release.

15.1.3.1. EXPLICIT RELEASE

B nodes explicitly release a name by broadcasting a notice.

P nodes send a notification to their NBNS.

M nodes both broadcast a notice and inform their supporting NBNS.

15.1.3.2. NAME LIFETIME AND REFRESH

Names held by an NBNS are given a lifetime during name registration. The NBNS will consider a name to have been silently released if the end-node fails to send a name refresh message to the NBNS before the lifetime expires. A refresh restarts the lifetime clock.

NOTE: The implementor should be aware of the tradeoff between accuracy of the database and the internet overhead that the refresh mechanism introduces. The lifetime period should be tuned accordingly.

For group names, each end-node must send refresh messages. A node that fails to do so will be considered to have silently released the name and dropped from the group.

The lifetime period is established through a simple negotiation mechanism during name registration: In the name registration request, the end-node proposes a lifetime value or requests an infinite lifetime. The NBNS places an actual lifetime value into the name registration response. The NBNS is always allowed to respond with an infinite actual period. If the end node proposed an infinite lifetime, the NBNS may respond with any definite period. If the end node proposed a definite period, the NBNS may respond with any definite period greater than or equal to that proposed.

This negotiation of refresh times gives the NBNS means to disable or enable refresh activity. The end-nodes may set a minimum refresh cycle period.

NBNS implementations which are completely reliable may disable refresh.

15.1.3.3. NAME CHALLENGE

To detect whether a node has silently released its claim to a name, it is necessary on occasion to challenge that node's current ownership. If the node defends the name then the node is allowed to continue possession. Otherwise it is assumed that the node has released the name.

A name challenge may be issued by an NBNS or by a P or M node. A challenge may be directed towards any end-node type: B, P, or M.

15.1.3.4. GROUP NAME FADE-OUT

NetBIOS groups may contain an arbitrarily large number of members. The time to challenge all members could be quite large.

To avoid long delays when names are claimed through an NBNS, an

optimistic heuristic has been adopted. It is assumed that there will always be some node which will defend a group name. Consequently, it is recommended that the NBNS will immediately reject a claim request for a unique name when there already exists a group with the same name. The NBNS will never return an IP address (in response to a NAME REGISTRATION REQUEST) when a group name exists.

An NBNS will consider a group to have faded out of existence when the last remaining member fails to send a timely refresh message or explicitly releases the name.

15.1.3.5. NAME CONFLICT

Name conflict exists when a unique name has been claimed by more than one node on a NetBIOS network. B, M, and NBNS nodes may detect a name conflict. The detection mechanism used by B and M nodes is active only during name discovery. The NBNS may detect conflict at any time it verifies the consistency of its name database.

B and M nodes detect conflict by examining the responses received in answer to a broadcast name query request. The first response is taken as authoritative. Any subsequent, inconsistent responses represent conflicts.

Subsequent responses are inconsistent with the authoritative response when:

```
    The subsequent response has the same transaction ID as the
    NAME QUERY REQUEST.
  AND
    The subsequent response is not a duplicate of the
    authoritative response.
  AND EITHER:
        The group/unique characteristic of the authoritative
        response is "unique".
      OR
        The group/unique characteristic of the subsequent
        response is "unique".
```

The period in which B and M nodes examine responses is limited by a conflict timer, CONFLICT_TIMER. The accuracy or duration of this timer is not crucial: the NetBIOS system will continue to operate even with persistent name conflicts.

Conflict conditions are signaled by sending a NAME CONFLICT DEMAND to the node owning the offending name. Nothing is sent to the node which originated the authoritative response.

Any end-node that receives NAME CONFLICT DEMAND is required to update its "local name table" to reflect that the name is in conflict. (The "local name table" on each node contains names that have been

successfully registered by that node.)

Notice that only those nodes that receive the name conflict message place a conflict mark next to a name.

Logically, a marked name does not exist on that node. This means that the node should not defend the name (for name claim purposes), should not respond to a name discovery requests for that name, nor should the node send name refresh messages for that name. Furthermore, it can no longer be used by that node for any session establishment or sending or receiving datagrams. Existing sessions are not affected at the time a name is marked as being in conflict.

The only valid user function against a marked name is DELETE NAME. Any other user NetBIOS function returns immediately with an error code of "NAME CONFLICT".

15.1.4. ADAPTER STATUS

An end-node or the NBNS may ask any other end-node for a collection of information about the NetBIOS status of that node. This status consists of, among other things, a list of the names which the node believes it owns. The returned status is filtered to contain only those names which have the same NetBIOS scope identifier as the requestor's name.

When requesting node status, the requestor identifies the target node by NetBIOS name A name query transaction may be necessary to acquire the IP address for the name. Locally cached name information may be used in lieu of a query transaction. The requesting node sends a NODE STATUS REQUEST. In response, the receiving node sends a NODE STATUS RESPONSE containing its local name table and various statistics.

The amount of status which may be returned is limited by the size of a UDP packet. However, this is sufficient for the typical NODE STATUS RESPONSE packet.

15.1.5. END-NODE NBNS INTERACTION

There are certain characteristics of end-node to NBNS interactions which are in common and are independent of any particular transaction type.

15.1.5.1. UDP, TCP, AND TRUNCATION

For all transactions between an end-node and an NBNS, either UDP or TCP may be used as a transport. If the NBNS receives a UDP based request, it will respond using UDP. If the amount of information exceeds what fits into a UDP packet, the response will contain a "truncation flag". In this situation, the end- node may open a TCP

connection to the NBNS, repeat the request, and receive a complete, untruncated response.

15.1.5.2. NBNS WACK

While a name service request is in progress, the NBNS may issue a WAIT FOR ACKNOWLEDGEMENT RESPONSE (WACK) to assure the client end-node that the NBNS is still operational and is working on the request.

15.1.5.3. NBNS REDIRECTION

The NBNS, because it follows Domain Name system styles of interaction, is permitted to redirect a client to another NBNS.

15.1.6. SECURED VERSUS NON-SECURED NBNS

An NBNS may be implemented in either of two general ways: The NBNS may monitor, and participate in, name activity to ensure consistency. This would be a "secured" style NBNS. Alternatively, an NBNS may be implemented to be essentially a "bulletin board" on which name information is posted and responsibility for consistency is delegated to the end-nodes. This would be a "non-secured" style NBNS.

15.1.7. CONSISTENCY OF THE NBNS DATA BASE

Even in a properly running NetBIOS scope the NBNS and its community of end-nodes may occasionally lose synchronization with respect to the true state of name registrations.

This may occur should the NBNS fail and lose all or part of its database.

More commonly, a P or M node may be turned-off (thus forgetting the names it has registered) and then be subsequently turned back on.

Finally, errors may occur or an implementation may be incorrect.

Various approaches have been incorporated into the NetBIOS-over- TCP protocols to minimize the impact of these problems.

1. The NBNS (or any other node) may "challenge" (using a NAME QUERY REQUEST) an end-node to verify that it actually owns a name.

   Such a challenge may occur at any time. Every end-node must be prepared to make a timely response.

   Failure to respond causes the NBNS to consider that the end-node has released the name in question.

(If UDP is being used as the underlying transport, the challenge, like all other requests, must be retransmitted some number of times in the absence of a response.)

2. The NBNS (or any other node) may request (using the NODE STATUS REQUEST) that an end-node deliver its entire name table.

   This may occur at any time. Every end-node must be prepared to make a timely response.

   Failure to respond permits (but does not require) the NBNS to consider that the end-node has failed and released all names to which it had claims. (Like the challenge, on a UDP transport, the request must be retransmitted in the absence of a response.)

3. The NBNS may revoke a P or M node's use of a name by sending either a NAME CONFLICT DEMAND or a NAME RELEASE REQUEST to the node.

   The receiving end-node may continue existing sessions which use that name, but must otherwise cease using that name. If the NBNS placed the name in conflict, the name may be re-acquired only by deletion and subsequent reclamation. If the NBNS requested that the name be released, the node may attempt to re-acquire the name without first performing a name release transaction.

4. The NBNS may impose a "time-to-live" on each name it registers. The registering node is made aware of this time value during the name registration procedure.

   Simple or reliable NBNS's may impose an infinite time-to-live.

5. If an end-node holds any names that have finite time-to-live values, then that node must periodically send a status report to the NBNS. Each name is reported using the NAME REFRESH REQUEST packet.

   These status reports restart the timers of both the NBNS and the reporting node. However, the only timers which are restarted are those associated with the name found in the status report. Timers on other names are not affected.

   The NBNS may consider that a node has released any name which has not been refreshed within some multiple of name's time-to-live.

   A well-behaved NBNS, would, however, issue a challenge to-,

or request a list of names from-, the non-reporting end-node before deleting its name(s). The absence of a response, or of the name in a response, will confirm the NBNS decision to delete a name.

6. The absence of reports may cause the NBNS to infer that the end-node has failed. Similarly, receipt of information widely divergent from what the NBNS believes about the node, may cause the NBNS to consider that the end-node has been restarted.

   The NBNS may analyze the situation through challenges or requests for a list of names.

7. A very cautious NBNS is free to poll nodes (by sending NAME QUERY REQUEST or NODE STATUS REQUEST packets) to verify that their name status is the same as that registered in the NBNS.

   NOTE: Such polling activity, if used at all by an implementation, should be kept at a very low level or enabled only during periods when the NBNS has some reason to suspect that its information base is inaccurate.

8. P and M nodes can detect incorrect name information at session establishment.

   If incorrect information is found, NBNS is informed via a NAME RELEASE REQUEST originated by the end-node which detects the error.

15.1.8. NAME CACHING

An end-node may keep a local cache of NetBIOS name-to-IP address translation entries.

All cache entries should be flushed on a periodic basis.

In addition, a node ought to flush any cache information associated with an IP address if the node receives any information indicating that there may be any possibility of trouble with the node at that IP address. For example, if a NAME CONFLICT DEMAND is sent to a node, all cached information about that node should be cleared within the sending node.

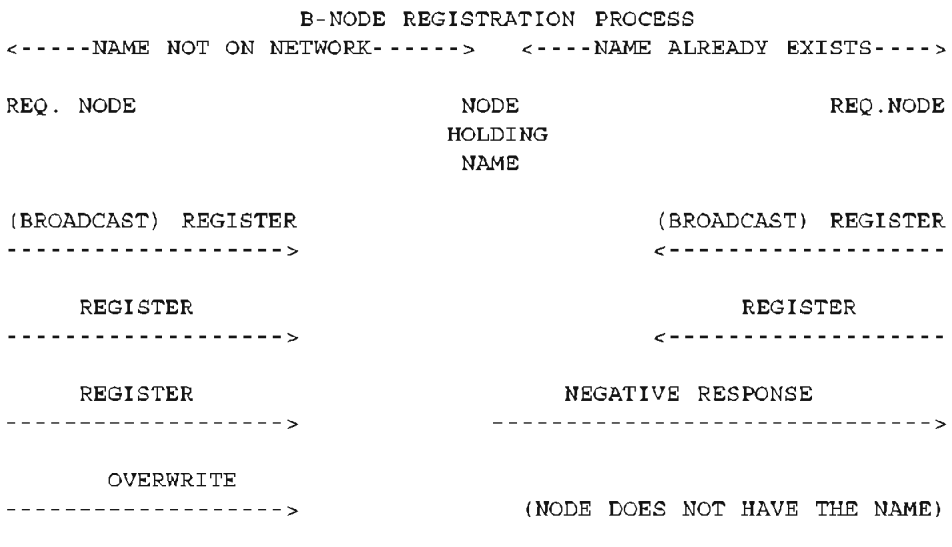
15.2. NAME REGISTRATION TRANSACTIONS

15.2.1. NAME REGISTRATION BY B NODES

A name claim transaction initiated by a B node is broadcast throughout the broadcast area. The NAME REGISTRATION REQUEST will be

heard by all B and M nodes in the area. Each node examines the claim to see whether it it is consistent with the names it owns. If an inconsistency exists, a NEGATIVE NAME REGISTRATION RESPONSE is unicast to the requestor. The requesting node obtains ownership of the name (or membership in the group) if, and only if, no NEGATIVE NAME REGISTRATION RESPONSEs are received within the name claim timeout, CONFLICT_TIMER. (See "Defined Constants and Variables" in the Detailed Specification for the value of this timer.)

A B node proclaims its new ownership by broadcasting a NAME OVERWRITE DEMAND.



The NAME REGISTRATION REQUEST, like any request, must be repeated if no response is received within BCAST_REQ_RETRY_TIMEOUT. Transmission of the request is attempted BCAST_REQ_RETRY_COUNT times.

15.2.2. NAME REGISTRATION BY P NODES

A name registration may proceed in various ways depending whether the name being registered is new to the NBNS. If the name is known to the NBNS, then challenges may be sent to the prior holder(s).

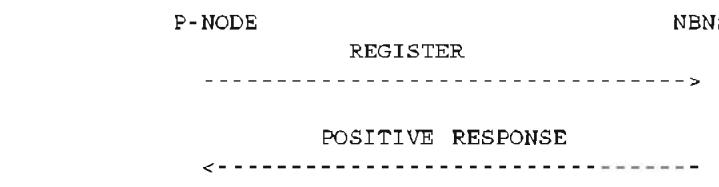
15.2.2.1. NEW NAME, OR NEW GROUP MEMBER

The diagram, below, shows the sequence of events when an end-node registers a name which is new to the NBNS. (The diagram omits WACKs, NBNS redirections, and retransmission of requests.)

This same interaction will occur if the name being registered is a group name and the group already exists. The NBNS will add the

registrant to the set of group members.

P-NODE REGISTRATION PROCESS
(server has no previous information about the name)



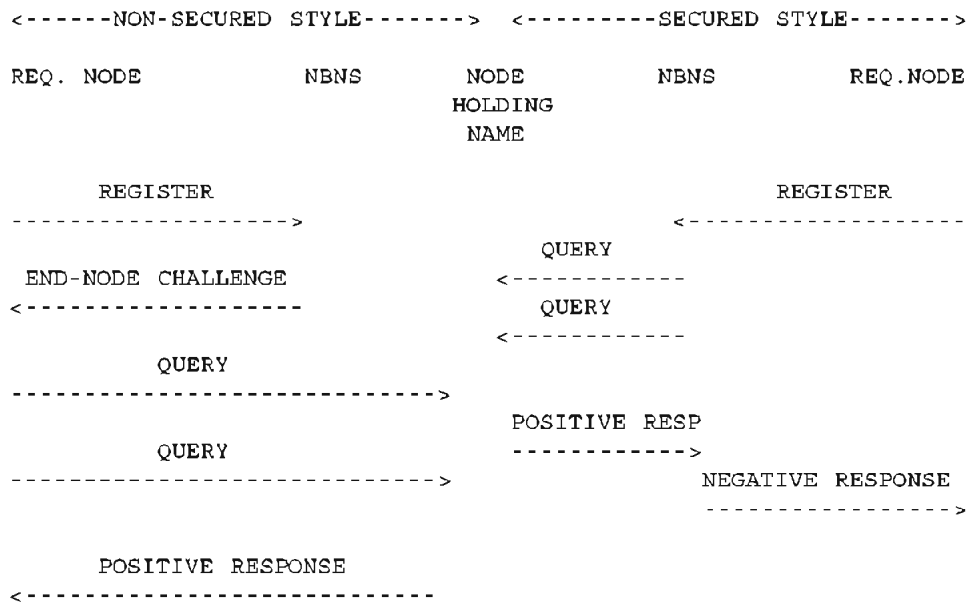
The interaction is rather simple: the end-node sends a NAME REGISTRATION REQUEST, the NBNS responds with a POSITIVE NAME REGISTRATION RESPONSE.

15.2.2.2. EXISTING NAME AND OWNER IS STILL ACTIVE

The following diagram shows interactions when an attempt is made to register a unique name, the NBNS is aware of an existing owner, and that existing owner is still active.

There are two sides to the diagram. The left side shows how a non-secured NBNS would handle the matter. Secured NBNS activity is shown on the right.

P-NODE REGISTRATION PROCESS
(server HAS a previous owner that IS active)



A non-secured NBNS will answer the NAME REGISTRATION REQUEST with a END-NODE CHALLENGE REGISTRATION RESPONSE. This response asks the end-node to issue a challenge transaction against the node indicated in the response. In this case, the prior node will defend against the challenge and the registering end-node will simply drop the registration attempt without further interaction with the NBNS.

A secured NBNS will refrain from answering the NAME REGISTRATION REQUEST until the NBNS has itself challenged the prior holder(s) of the name. In this case, the NBNS finds that that the name is still being defended and consequently returns a NEGATIVE NAME REGISTRATION RESPONSE to the registrant.

Due to the potential time for the secured NBNS to make the challenge(s), it is likely that a WACK will be sent by the NBNS to the registrant.

Although not shown in the diagram, a non-secured NBNS will send a NEGATIVE NAME REGISTRATION RESPONSE to a request to register a unique name when there already exists a group of the same name. A secured NBNS may elect to poll (or challenge) the group members to determine whether any active members remain. This may impose a heavy load on the network. It is recommended that group names be allowed to fade-out through the name refresh mechanism.

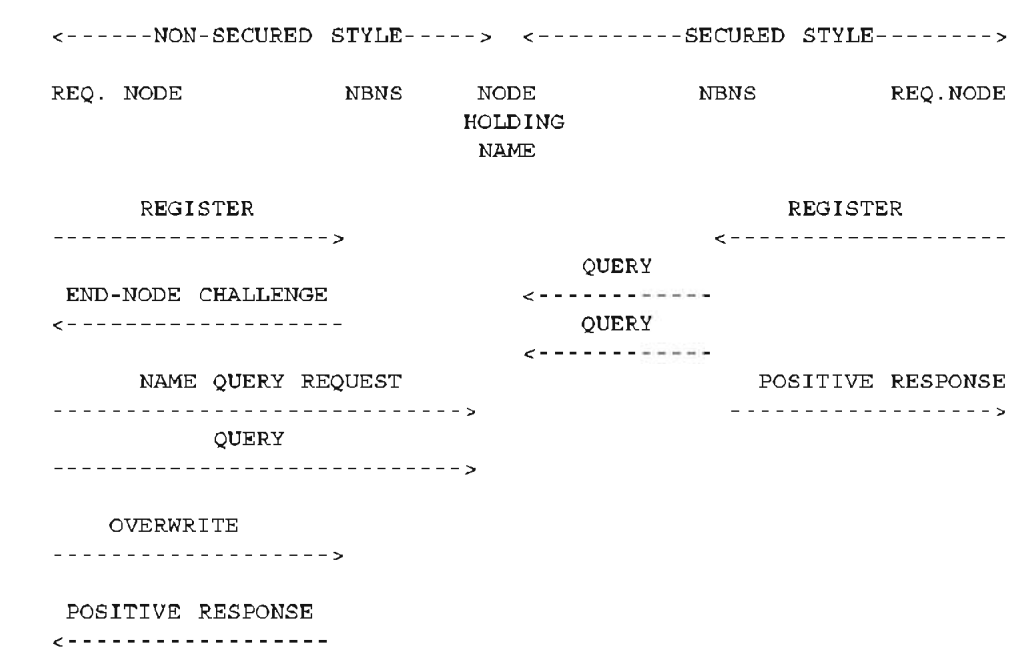
15.2.2.3. EXISTING NAME AND OWNER IS INACTIVE

The following diagram shows interactions when an attempt is made to register a unique name, the NBNS is aware of an existing owner, and that existing owner is no longer active.

A non-secured NBNS will answer the NAME REGISTRATION REQUEST with a END-NODE CHALLENGE REGISTRATION RESPONSE. This response asks the end-node to issue a challenge transaction against the node indicated in the response. In this case, the prior node will not defend against the challenge. The registrant will inform the NBNS through a NAME OVERWRITE REQUEST. The NBNS will replace the prior name information in its database with the information in the overwrite request.

A secured NBNS will refrain from answering the NAME REGISTRATION REQUEST until the NBNS has itself challenged the prior holder(s) of the name. In this case, the NBNS finds that that the name is not being defended and consequently returns a POSITIVE NAME REGISTRATION RESPONSE to the registrant.

P-NODE REGISTRATION PROCESS
(server HAS a previous owner that is NOT active)



Due to the potential time for the secured NBNS to make the challenge(s), it is likely that a WACK will be sent by the NBNS to the registrant.

A secured NBNS will immediately send a NEGATIVE NAME REGISTRATION RESPONSE in answer to any NAME OVERWRITE REQUESTS it may receive.

15.2.3. NAME REGISTRATION BY M NODES

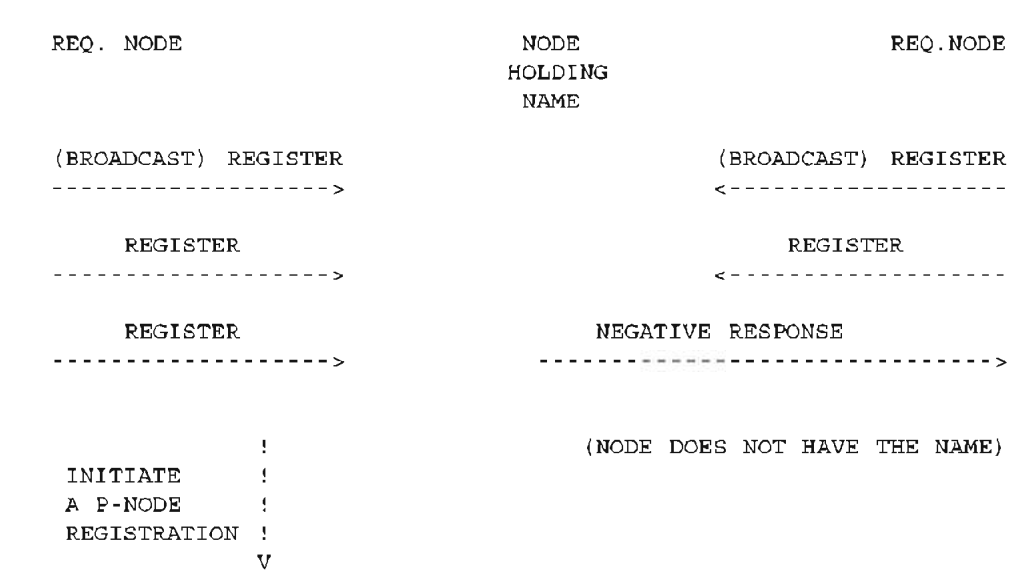
An M node begin a name claim operation as if the node were a B node: it broadcasts a NAME REGISTRATION REQUEST and listens for NEGATIVE NAME REGISTRATION RESPONSEs. Any NEGATIVE NAME REGISTRATION RESPONSE prevents the M node from obtaining the name and terminates the claim operation.

If, however, the M node does not receive any NEGATIVE NAME REGISTRATION RESPONSE, the M node must continue the claim procedure as if the M node were a P node.

Only if both name claims were successful does the M node acquire the name.

The following diagram illustrates M node name registration:

M-NODE REGISTRATION PROCESS



15.3. NAME QUERY TRANSACTIONS

Name query transactions are initiated by end-nodes to obtain the IP address(es) and other attributes associated with a NetBIOS name.

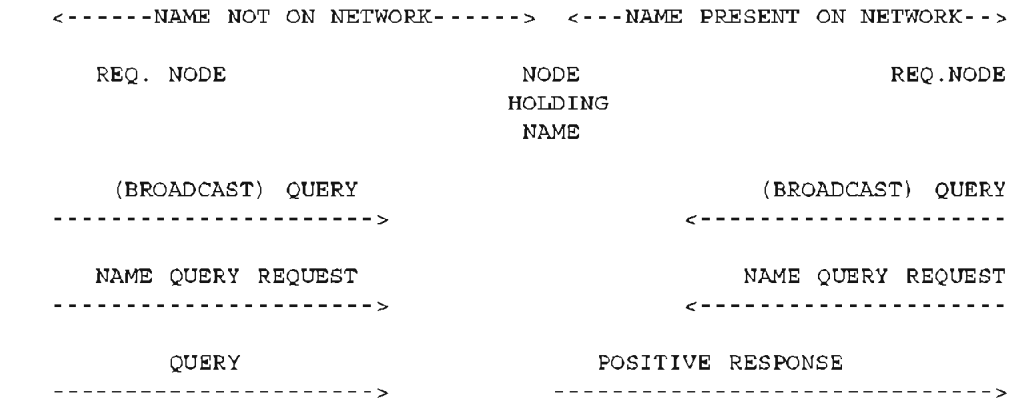
15.3.1. QUERY BY B NODES

The following diagram shows how B nodes go about discovering who owns a name.

The left half of the diagram illustrates what happens if there are no holders of the name. In that case no responses are received in answer to the broadcast NAME QUERY REQUEST(s).

The right half shows a POSITIVE NAME QUERY RESPONSE unicast by a name holder in answer to the broadcast request. A name holder will make this response to every NAME QUERY REQUEST that it hears. And each holder acts this way. Thus, the node sending the request may receive many responses, some duplicates, and from many nodes.

B-NODE DISCOVERY PROCESS



Name query is generally, but not necessarily, a prelude to NetBIOS session establishment or NetBIOS datagram transmission. However, name query may be used for other purposes.

A B node may elect to build a group membership list for subsequent use (e.g. for session establishment) by collecting and saving the responses.

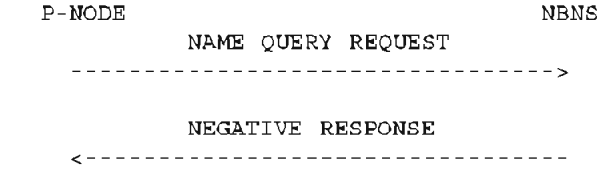
15.3.2. QUERY BY P NODES

An NBNS answers queries from a P node with a list of IP address and other information for each owner of the name. If there are multiple owners (i.e. if the name is a group name), the NBNS loads as many answers into the response as will fit into a UDP packet. A truncation flag indicates whether any additional owner information remains. All the information may be obtained by repeating the query over a TCP connection.

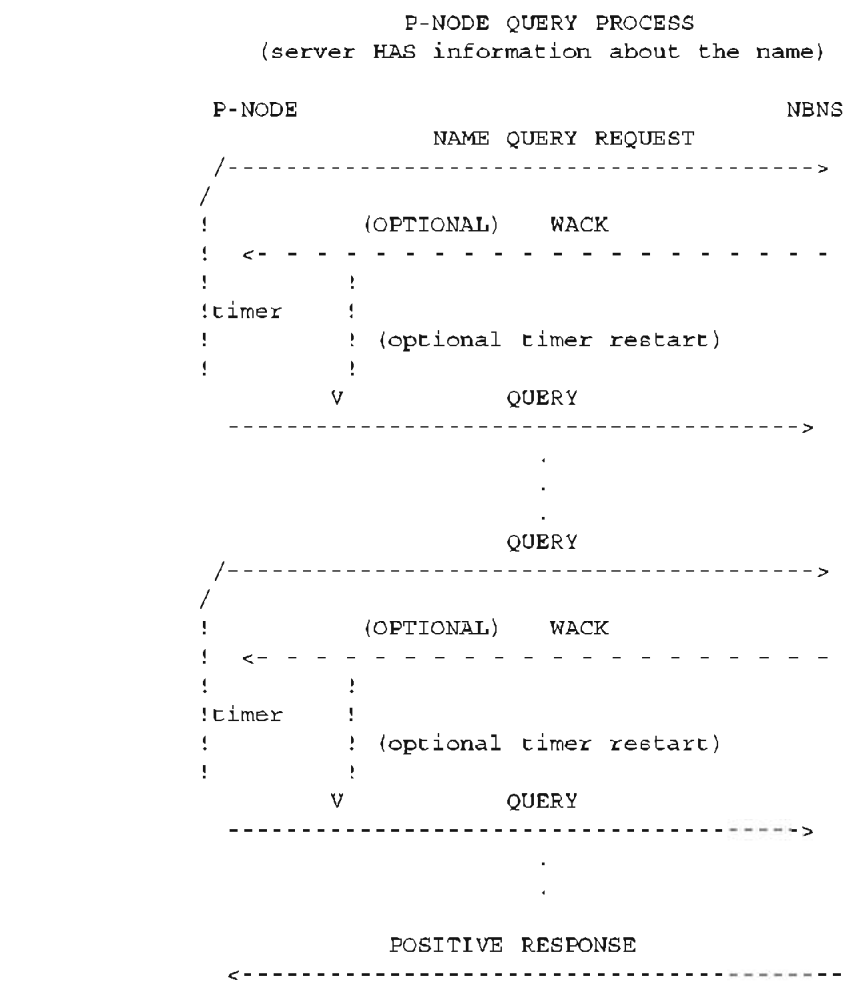
The NBNS is not required to impose any order on its answer list.

The following diagram shows what happens if the NBNS has no information about the name:

P-NODE DISCOVERY PROCESS
(server has no information about the name)

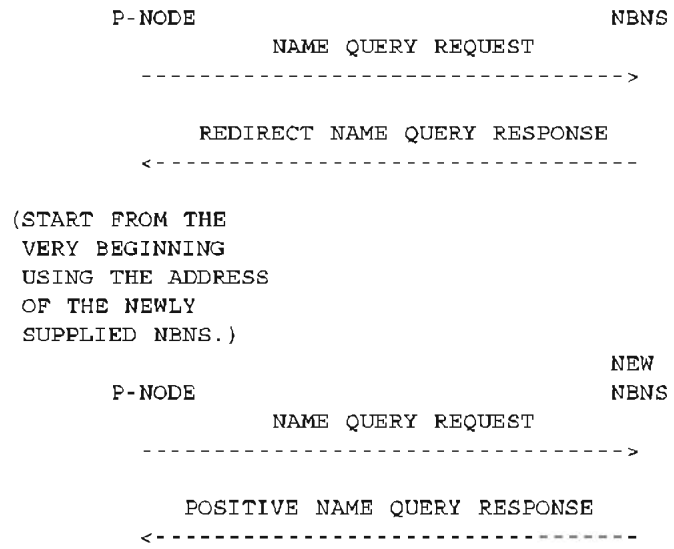


The next diagram illustrates interaction between the end-node and the NBNS when the NBNS does have information about the name. This diagram shows, in addition, the retransmission of the request by the end-node in the absence of a timely response. Also shown are WACKs (or temporary, intermediate responses) sent by the NBNS to the end-node:



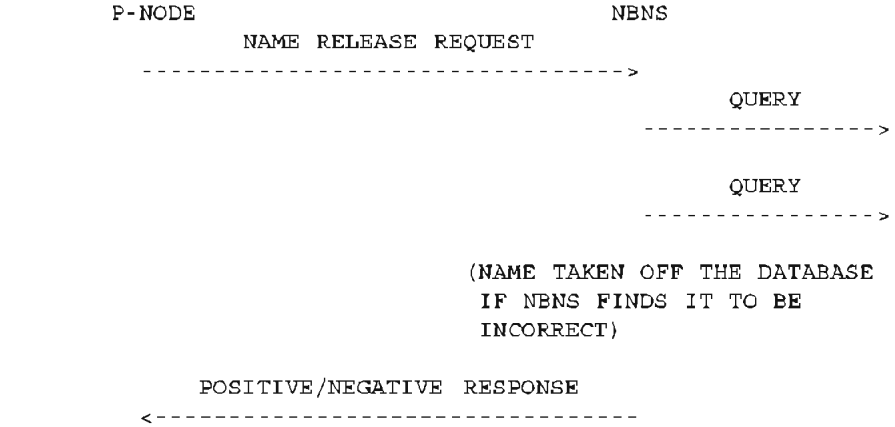
The following diagram illustrates NBNS redirection. Upon receipt of a NAME QUERY REQUEST, the NBNS redirects the client to another NBNS. The client repeats the request to the new NBNS and obtains a response. The diagram shows that response as a POSITIVE NAME QUERY RESPONSE. However any legal NBNS response may occur in actual operation.

NBNS REDIRECTION



The next diagram shows how a P or M node tells the NBNS that the NBNS has provided incorrect information. This procedure may begin after a DATAGRAM ERROR packet has been received or a session set-up attempt has discovered that the NetBIOS name does not exist at the destination, the IP address of which was obtained from the NBNS during a prior name query transaction. The NBNS, in this case a secure NBNS, issues queries to verify whether the information is, in fact, incorrect. The NBNS closes the transaction by sending either a POSITIVE or NEGATIVE NAME RELEASE RESPONSE, depending on the results of the verification.

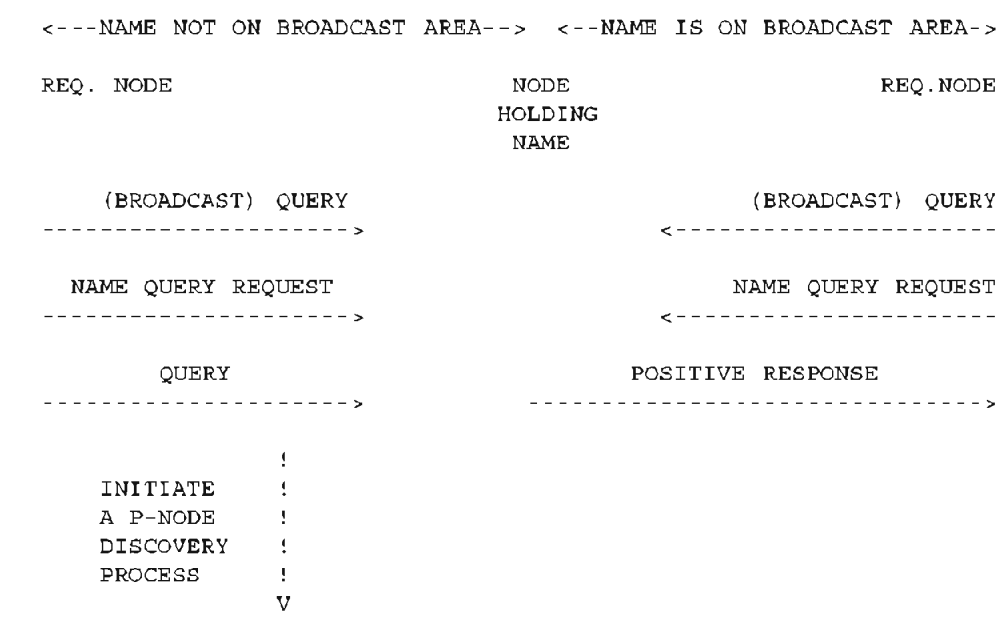
CORRECTING NBNS INFORMATION BASE



15.3.3. QUERY BY M NODES

M node name query follows the B node pattern. In the absence of adequate results, the M node then continues by performing a P node type query. This is shown in the following diagram:

M-NODE DISCOVERY PROCESS



15.3.4. ACQUIRE GROUP MEMBERSHIP LIST

The entire membership of a group may be acquired by sending a NAME QUERY REQUEST to the NBNS. The NBNS will respond with a POSITIVE NAME QUERY RESPONSE or a NEGATIVE NAME QUERY RESPONSE. A negative response completes the procedure and indicates that there are no members in the group.

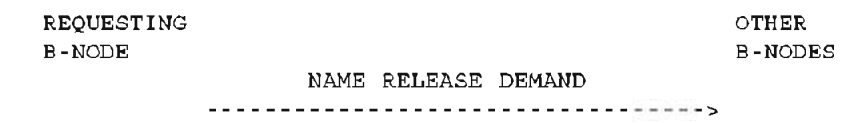
If the positive response has the truncation bit clear, then the response contains the entire list of group members. If the truncation bit is set, then this entire procedure must be repeated, but using TCP as a foundation rather than UDP.

15.4. NAME RELEASE TRANSACTIONS

15.4.1. RELEASE BY B NODES

A NAME RELEASE DEMAND contains the following information:

- NetBIOS name
- The scope of the NetBIOS name
- Name type: unique or group
- IP address of the releasing node
- Transaction ID



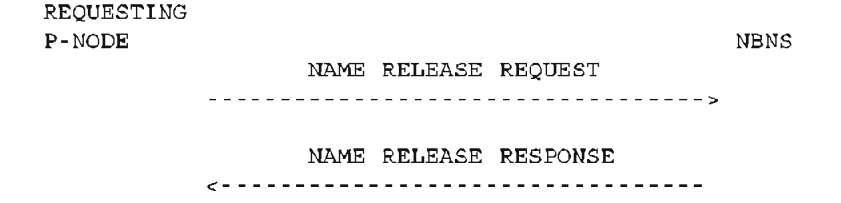
15.4.2. RELEASE BY P NODES

A NAME RELEASE REQUEST contains the following information:

- NetBIOS name
- The scope of the NetBIOS name
- Name type: unique or group
- IP address of the releasing node
- Transaction ID

A NAME RELEASE RESPONSE contains the following information:

- NetBIOS name
- The scope of the NetBIOS name
- Name type: unique or group
- IP address of the releasing node
- Transaction ID
- Result:
  - Yes: name was released
  - No: name was not released, a reason code is provided



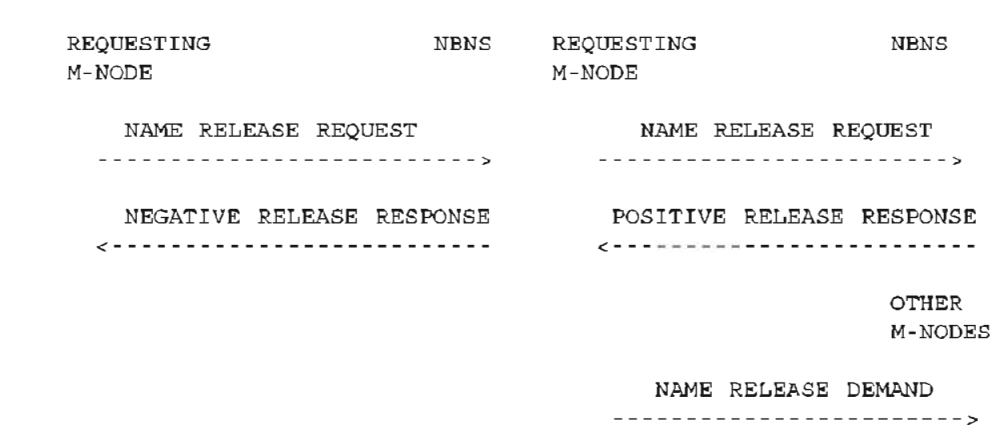
15.4.3. RELEASE BY M NODES

The name release procedure of the M node is a combination of the P and B node name release procedures. The M node first performs the P

release procedure. If the P procedure fails then the release procedure does not continue, it fails. If and only if the P procedure succeeds then the M node broadcasts the NAME RELEASE DEMAND to the broadcast area, the B procedure.

NOTE: An M node typically performs a B-style operation and then a P-style operation. In this case, however, the P-style operation comes first.

The following diagram illustrates the M node name release procedure:



15.5. NAME MAINTENANCE TRANSACTIONS

15.5.1. NAME REFRESH

Name refresh transactions are used to handle the following situations:

a) An NBNS node needs to detect if a P or M node has "silently" gone down, so that names held by that node can be purged from the data base.

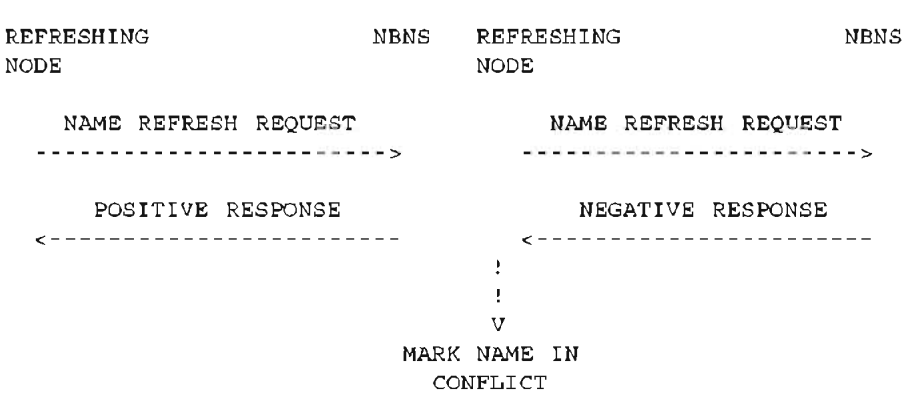
b) If the NBNS goes down, it needs to be able to reconstruct the data base when it comes back up.

c) If the network should be partitioned, the NBNS needs to be able to able to update its data base when the network reconnects.

Each P or M node is responsible for sending periodic NAME REFRESH REQUESTs for each name that it has registered. Each refresh packet contains a single name that has been successfully registered by that

node. The interval between such packets is negotiated between the end node and the NBNS server at the time that the name is initially claimed. At name claim time, an end node will suggest a refresh timeout value. The NBNS node can modify this value in the reply packet. A NBNS node can also choose to tell the end node to not send any refresh packet by using the "infinite" timeout value in the response packet. The timeout value returned by the NBNS is the actual refresh timeout that the end node must use.

When a node sends a NAME REFRESH REQUEST, it must be prepared to receive a negative response. This would happen, for example, if the the NBNS discovers that the the name had already been assigned to some other node. If such a response is received, the end node should mark the name as being in conflict. Such an entry should be treated in the same way as if name conflict had been detected against the name. The following diagram illustrates name refresh:



15.5.2. NAME CHALLENGE

Name challenge is done by sending a NAME QUERY REQUEST to an end node of any type. If a POSITIVE NAME QUERY RESPONSE is returned, then that node still owns the name. If a NEGATIVE NAME QUERY RESPONSE is received or if no response is received, it can be assumed that the end node no longer owns the name.

Name challenge can be performed either by the NBNS node, or by an end node. When an end-node sends a name claim packet, the NBNS node may do the challenge operation. The NBNS node can choose, however, to require the end node do the challenge. In that case, the NBNS will send an END-NODE CHALLENGE RESPONSE packet to the end node, which should then proceed to challenge the putative owner.

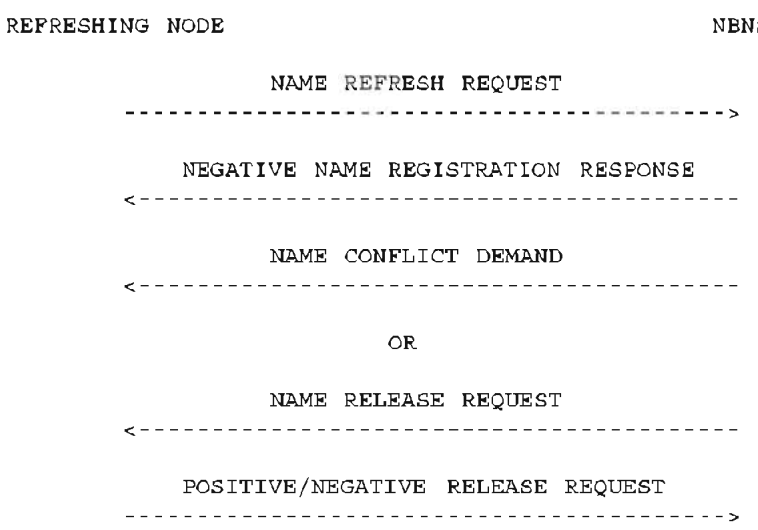
Note that the name challenge procedure sends a normal NAME QUERY REQUEST packet to the end node. It does not require a special packet. The only new packet introduced is the END-NODE CHALLENGE

RESPONSE which is sent by an NBNS node when the NBNS wants the end-node to perform the challenge operation.

15.5.3. CLEAR NAME CONFLICT

It is possible during a refresh request from a M or P node for a NBNS to detects a name in conflict. The response to the NAME REFRESH REQUEST must be a NEGATIVE NAME REGISTRATION RESPONSE. Optionally, in addition, the NBNS may send a NAME CONFLICT DEMAND or a NAME RELEASE REQUEST to the refreshing node. The NAME CONFLICT DEMAND forces the node to place the name in the conflict state. The node will eventually inform it's user of the conflict. The NAME RELEASE REQUEST will force the node to flush the name from its local name table completely. This forces the node to flush the name in conflict. This does not cause termination of existing sessions using this name.

The following diagram shows an NBNS detecting and correcting a conflict:



15.6. ADAPTER STATUS TRANSACTIONS

Adapter status is obtained from a node as follows:

1. Perform a name discovery operation to obtain the IP addresses of a set of end-nodes.

2. Repeat until all end-nodes from the set have been used:

   a. Select one end-node from the set.

   b. Send a NODE STATUS REQUEST to that end-node using UDP.

c. Await a NODE STATUS RESPONSE. (If a timely response is not forthcoming, repeat step "b" UCAST_REQ_RETRY_COUNT times. After the last retry, go to step "a".)

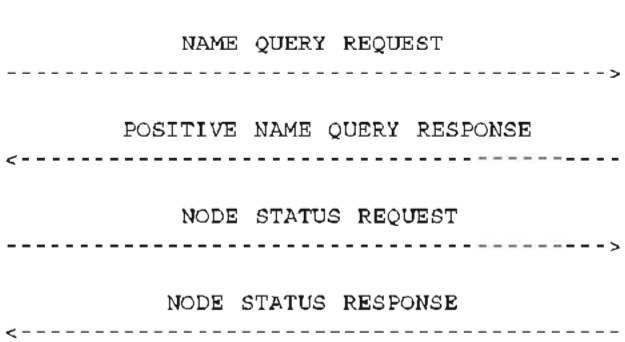
d. If the truncation bit is not set in the response, the response contains the entire node status. Return the status to the user and terminate this procedure.

e. If the truncation bit is set in the response, then not all status was returned because it would not fit into the response packet. The responder will set the truncation bit if the IP datagram length would exceed MAX_DATAGRAM_LENGTH. Return the status to the user and terminate this procedure.

3. Return error to user, no status obtained.

The repetition of step 2, above, through all nodes of the set, is optional.

Following is an example transaction of a successful Adapter Status operation:



16. NetBIOS SESSION SERVICE

The NetBIOS session service begins after one or more IP addresses have been found for the target name. These addresses may have been acquired using the NetBIOS name query transactions or by other means, such as a local name table or cache.

NetBIOS session service transactions, packets, and protocols are identical for all end-node types. They involve only directed (point-to-point) communications.

16.1. OVERVIEW OF NetBIOS SESSION SERVICE

Session service has three phases:

Session establishment - it is during this phase that the IP address and TCP port of the called name is determined, and a TCP connection is established with the remote party.

Steady state - it is during this phase that NetBIOS data messages are exchanged over the session. Keep-alive packets may also be exchanged if the participating nodes are so configured.

Session close - a session is closed whenever either a party (in the session) closes the session or it is determined that one of the parties has gone down.

16.1.1. SESSION ESTABLISHMENT PHASE OVERVIEW

An end-node begins establishment of a session to another node by somehow acquiring (perhaps using the name query transactions or a local cache) the IP address of the node or nodes purported to own the destination name.

Every end-node awaits incoming NetBIOS session requests by listening for TCP calls to a well-known service port, SSN_SRVC_TCP_PORT. Each incoming TCP connection represents the start of a separate NetBIOS session initiation attempt. The NetBIOS session server, not the ultimate application, accepts the incoming TCP connection(s).

Once the TCP connection is open, the calling node sends session service request packet. This packet contains the following information:

- Calling IP address (see note)
- Calling NetBIOS name
- Called IP address (see note)
- Called NetBIOS name

NOTE: The IP addresses are obtained from the TCP service interface.

When the session service request packet arrives at the NetBIOS server, one of the the following situations will exist:

- There exists a NetBIOS LISTEN compatible with the incoming call and there are adequate resources to permit session establishment to proceed.

- There exists a NetBIOS LISTEN compatible with the incoming call, but there are inadequate resources to permit

establishment of a session.

- The called name does, in fact, exist on the called node, but there is no pending NetBIOS LISTEN compatible with the incoming call.

- The called name does not exist on the called node.

In all but the first case, a rejection response is sent back over the TCP connection to the caller. The TCP connection is then closed and the session phase terminates. Any retry is the responsibility of the caller. For retries in the case of a group name, the caller may use the next member of the group rather than immediately retrying the instant address. In the case of a unique name, the caller may attempt an immediate retry using the same target IP address unless the called name did not exist on the called node. In that one case, the NetBIOS name should be re-resolved.

If a compatible LISTEN exists, and there are adequate resources, then the session server may transform the existing TCP connection into the NetBIOS data session. Alternatively, the session server may redirect, or "retarget" the caller to another TCP port (and IP address).

If the caller is redirected, the caller begins the session establishment anew, but using the new IP address and TCP port given in the retarget response. Again a TCP connection is created, and again the calling and called node exchange credentials. The called party may accept the call, reject the call, or make a further redirection.

This mechanism is based on the presumption that, on hosts where it is not possible to transfer open TCP connections between processes, the host will have a central session server. Applications willing to receive NetBIOS calls will obtain an ephemeral TCP port number, post a TCP unspecified passive open on that port, and then pass that port number and NetBIOS name information to the NetBIOS session server using a NetBIOS LISTEN operation. When the call is placed, the session server will "retarget" the caller to the application's TCP socket. The caller will then place a new call, directly to the application. The application has the responsibility to mimic the session server at least to the extent of receiving the calling credentials and then accepting or rejecting the call.

16.1.1.1. RETRYING AFTER BEING RETARGETTED

A calling node may find that it can not establish a session with a node to which it was directed by the retargetting procedure. Since retargetting may be nested, there is an issue whether the caller should begin a retry at the initial starting point or back-up to an intermediate retargetting point. The caller may use any method. A

discussion of two such methods is in Appendix B, "Retarget Algorithms".

16.1.1.2. SESSION ESTABLISHMENT TO A GROUP NAME

Session establishment with a group name requires special consideration. When a NetBIOS CALL attempt is made to a group name, name discovery will result in a list (possibly incomplete) of the members of that group. The calling node selects one member from the list and attempts to build a session. If that fails, the calling node may select another member and make another attempt.

When the session service attempts to make a connection with one of the members of the group, there is no guarantee that that member has a LISTEN pending against that group name, that the called node even owns, or even that the called node is operating.

16.1.2. STEADY STATE PHASE OVERVIEW

NetBIOS data messages are exchanged in the steady state. NetBIOS messages are sent by prepending the user data with a message header and sending the header and the user data over the TCP connection. The receiver removes the header and passes the data to the NetBIOS user.

In order to detect failure of one of the nodes or of the intervening network, "session keep alive" packets may be periodically sent in the steady state.

Any failure of the underlying TCP connection, whether a reset, a timeout, or other failure, implies failure of the NetBIOS session.

16.1.3. SESSION TERMINATION PHASE OVERVIEW

A NetBIOS session is terminated normally when the user requests the session to be closed or when the session service detects the remote partner of the session has gracefully terminated the TCP connection. A NetBIOS session is abnormally terminated when the session service detects a loss of the connection. Connection loss can be detected with the keep-alive function of the session service or TCP, or on the failure of a SESSION MESSAGE send operation.

When a user requests to close a session, the service first attempts a graceful in-band close of the TCP connection. If the connection does not close within the SSN_CLOSE_TIMEOUT the TCP connection is aborted. No matter how the TCP connection is terminated, the NetBIOS session service always closes the NetBIOS session.

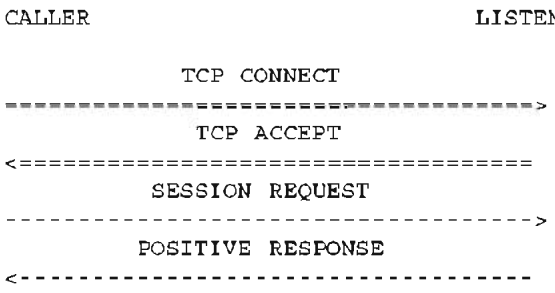
When the session service receives an indication from TCP that a connection close request has been received, the TCP connection and the NetBIOS session are immediately closed and the user is informed

of the loss of the session. All data received up to the close indication should be delivered, if possible, to the session's user.

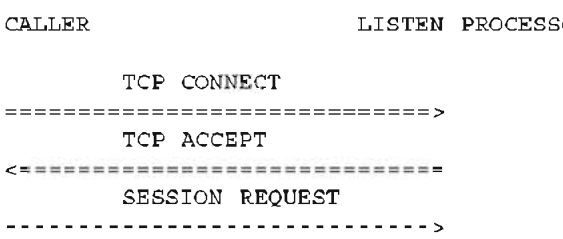
16.2. SESSION ESTABLISHMENT PHASE

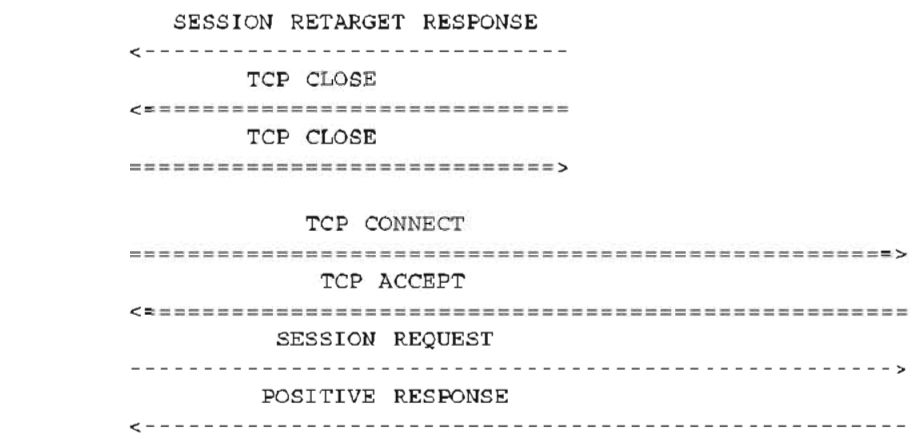
All the following diagrams assume a name query operation was successfully completed by the caller node for the listener's name.

This first diagram shows the sequence of network events used to successfully establish a session without retargetting by the listener. The TCP connection is first established with the well-known NetBIOS session service TCP port, SSN_SRVC_TCP_PORT. The caller then sends a SESSION REQUEST packet over the TCP connection requesting a session with the listener. The SESSION REQUEST contains the caller's name and the listener's name. The listener responds with a POSITIVE SESSION RESPONSE informing the caller this TCP connection is accepted as the connection for the data transfer phase of the session.

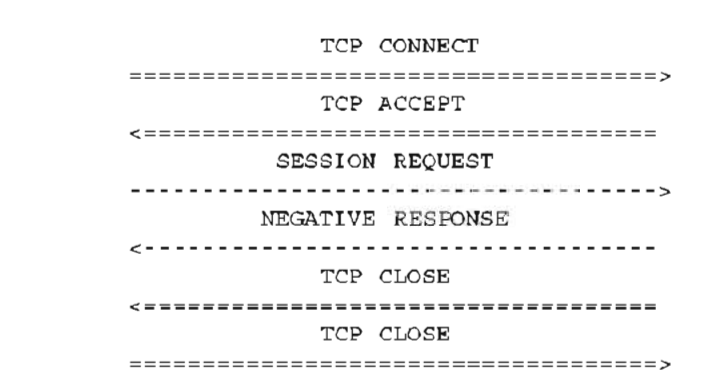


The second diagram shows the sequence of network events used to successfully establish a session when the listener does retargetting. The session establishment procedure is the same as with the first diagram up to the listener's response to the SESSION REQUEST. The listener, divided into two sections, the listen processor and the actual listener, sends a SESSION RETARGET RESPONSE to the caller. This response states the call is acceptable, but the data transfer TCP connection must be at the new IP address and TCP port. The caller then re-iterates the session establishment process anew with the new IP address and TCP port after the initial TCP connection is closed. The new listener then accepts this connection for the data transfer phase with a POSITIVE SESSION RESPONSE.

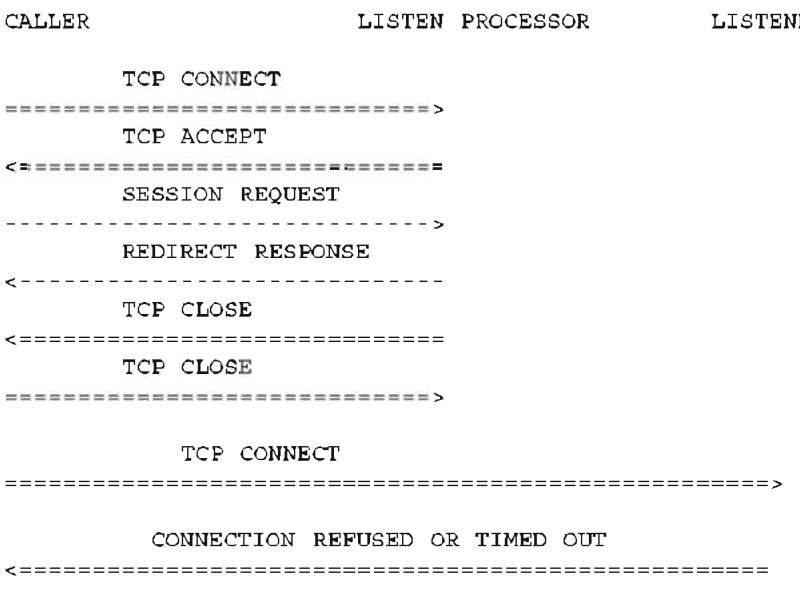




The third diagram is the sequence of network events for a rejected session request with the listener. This type of rejection could occur with either a non-retargetting listener or a retargetting listener. After the TCP connection is established at SSN_SRVC_TCP_PORT, the caller sends the SESSION REQUEST over the TCP connection. The listener does not have either a listen pending for the listener's name or the pending NetBIOS listen is specific to another caller's name. Consequently, the listener sends a NEGATIVE SESSION RESPONSE and closes the TCP connection.



The fourth diagram is the sequence of network events when session establishment fails with a retargetting listener. After being redirected, and after the initial TCP connection is closed the caller tries to establish a TCP connection with the new IP address and TCP port. The connection fails because either the port is unavailable or the target node is not active. The port unavailable race condition occurs if another caller has already acquired the TCP connection with the listener. For additional implementation suggestions, see Appendix B, "Retarget Algorithms".



16.3. SESSION DATA TRANSFER PHASE

16.3.1. DATA ENCAPSULATION

NetBIOS messages are exchanged in the steady state. Messages are sent by prepending user data with message header and sending the header and the user data over the TCP connection. The receiver removes the header and delivers the NetBIOS data to the user.

16.3.2. SESSION KEEP-ALIVES

In order to detect node failure or network partitioning, "session keep alive" packets are periodically sent in the steady state. A session keep alive packet is discarded by a peer node.

A session keep alive timer is maintained for each session. This timer is reset whenever any data is sent to, or received from, the session peer. When the timer expires, a NetBIOS session keep-alive packet is sent on the TCP connection. Sending the keep-alive packet forces data to flow on the TCP connection, thus indirectly causing TCP to detect whether the connection is still active.

Since many TCP implementations provide a parallel TCP "keep- alive" mechanism, the NetBIOS session keep-alive is made a configurable option. It is recommended that the NetBIOS keep- alive mechanism be used only in the absence of TCP keep-alive.

Note that unlike TCP keep alives, NetBIOS session keep alives do not require a response from the NetBIOS peer -- the fact that it was

possible to send the NetBIOS session keep alive is sufficient indication that the peer, and the connection to it, are still active.

The only requirement for interoperability is that when a session keep alive packet is received, it should be discarded.

17. NETBIOS DATAGRAM SERVICE

17.1. OVERVIEW OF NetBIOS DATAGRAM SERVICE

Every NetBIOS datagram has a named destination and source. To transmit a NetBIOS datagram, the datagram service must perform a name query operation to learn the IP address and the attributes of the destination NetBIOS name. (This information may be cached to avoid the overhead of name query on subsequent NetBIOS datagrams.)

NetBIOS datagrams are carried within UDP packets. If a NetBIOS datagram is larger than a single UDP packet, it may be fragmented into several UDP packets.

End-nodes may receive NetBIOS datagrams addressed to names not held by the receiving node. Such datagrams should be discarded. If the name is unique then a DATAGRAM ERROR packet is sent to the source of that NetBIOS datagram.

17.1.1. UNICAST, MULTICAST, AND BROADCAST

NetBIOS datagrams may be unicast, multicast, or broadcast. A NetBIOS datagram addressed to a unique NetBIOS name is unicast. A NetBIOS datatgram addressed to a group NetBIOS name, whether there are zero, one, or more actual members, is multicast. A NetBIOS datagram sent using the NetBIOS "Send Broadcast Datagram" primitive is broadcast.

17.1.2. FRAGMENTATION OF NetBIOS DATAGRAMS

When the header and data of a NetBIOS datagram exceeds the maximum amount of data allowed in a UDP packet, the NetBIOS datagram must be fragmented before transmission and reassembled upon receipt.

A NetBIOS Datagram is composed of the following protocol elements:

- IP header of 20 bytes (minimum)
- UDP header of 8 bytes
- NetBIOS Datagram Header of 14 bytes
- The NetBIOS Datagram data.

The NetBIOS Datagram data section is composed of 3 parts:

- Source NetBIOS name (255 bytes maximum)
- Destination NetBIOS name (255 bytes maximum)
- The NetBIOS user's data (maximum of 512 bytes)

The two name fields are in second level encoded format (see section 14.)

A maximum size NetBIOS datagram is 1064 bytes. The minimal maximum IP datagram size is 576 bytes. Consequently, a NetBIOS Datagram may not fit into a single IP datagram. This makes it necessary to permit the fragmentation of NetBIOS Datagrams.

On networks meeting or exceeding the minimum IP datagram length requirement of 576 octets, at most two NetBIOS datagram fragments will be generated. The protocols and packet formats accommodate fragmentation into three or more parts.

When a NetBIOS datagram is fragmented, the IP, UDP and NetBIOS Datagram headers are present in each fragment. The NetBIOS Datagram data section is split among resulting UDP datagrams. The data sections of NetBIOS datagram fragments do not overlap. The only fields of the NetBIOS Datagram header that would vary are the FLAGS and OFFSET fields.

The FIRST bit in the FLAGS field indicate whether the fragment is the first in a sequence of fragments. The MORE bit in the FLAGS field indicates whether other fragments follow.

The OFFSET field is the byte offset from the beginning of the NetBIOS datagram data section to the first byte of the data section in a fragment. It is 0 for the first fragment. For each subsequent fragment, OFFSET is the sum of the bytes in the NetBIOS data sections of all preceding fragments.

If the NetBIOS datagram was not fragmented:

- FIRST = TRUE
- MORE = FALSE
- OFFSET = 0

If the NetBIOS datagram was fragmented:

- First fragment:
  - FIRST = TRUE
  - MORE = TRUE
  - OFFSET = 0
- Intermediate fragments:
  - FIRST = FALSE
  - MORE = TRUE
  - OFFSET = sum(NetBIOS data in prior fragments)
- Last fragment:
  - FIRST = FALSE
  - MORE = FALSE

- OFFSET = sum(NetBIOS data in prior fragments)

The relative position of intermediate fragments may be ascertained from OFFSET.

An NBDD must remember the destination name of the first fragment in order to relay the subsequent fragments of a single NetBIOS datagram. The name information can be associated with the subsequent fragments through the transaction ID, DGM_ID, and the SOURCE_IP, fields of the packet. This information can be purged by the NBDD after the last fragment has been processed or FRAGMENT_TO time has expired since the first fragment was received.

17.2. NetBIOS DATAGRAMS BY B NODES

For NetBIOS datagrams with a named destination (i.e. non- broadcast), a B node performs a name discovery for the destination name before sending the datagram. (Name discovery may be bypassed if information from a previous discovery is held in a cache.) If the name type returned by name discovery is UNIQUE, the datagram is unicast to the sole owner of the name. If the name type is GROUP, the datagram is broadcast to the entire broadcast area using the destination IP address BROADCAST_ADDRESS.

A receiving node always filters datagrams based on the destination name. If the destination name is not owned by the node or if no RECEIVE DATAGRAM user operations are pending for the name, then the datagram is discarded. For datagrams with a UNIQUE name destination, if the name is not owned by the node then the receiving node sends a DATAGRAM ERROR packet. The error packet originates from the DGM_SRVC_UDP_PORT and is addressed to the SOURCE_IP and SOURCE_PORT from the bad datagram. The receiving node quietly discards datagrams with a GROUP name destination if the name is not owned by the node.

Since broadcast NetBIOS datagrams do not have a named destination, the B node sends the DATAGRAM SERVICE packet(s) to the entire broadcast area using the destination IP address BROADCAST_ADDRESS. In order for the receiving nodes to distinguish this datagram as a broadcast NetBIOS datagram, the NetBIOS name used as the destination name is '*' (hexadecimal 2A) followed by 15 bytes of hexidecimal 00. The NetBIOS scope identifier is appended to the name before it is converted into second-level encoding. For example, if the scope identifier is "NETBIOS.SCOPE" then the first-level encoded name would be:

CKAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA.NETBIOS.SCOPE

According to [2], a user may not provide a NetBIOS name beginning with "*".

For each node in the broadcast area that receives the NetBIOS

broadcast datagram, if any RECEIVE BROADCAST DATAGRAM user operations are pending then the data from the NetBIOS datagram is replicated and delivered to each. If no such operations are pending then the node silently discards the datagram.

17.3. NetBIOS DATAGRAMS BY P AND M NODES

P and M nodes do not use IP broadcast to distribute NetBIOS datagrams.

Like B nodes, P and M nodes must perform a name discovery or use cached information to learn whether a destination name is a group or a unique name.

Datagrams to unique names are unicast directly to the destination by P and M nodes, exactly as they are by B nodes.

Datagrams to group names and NetBIOS broadcast datagrams are unicast to the NBDD. The NBDD then relays the datagrams to each of the nodes specified by the destination name.

An NBDD may not be capable of sending a NetBIOS datagram to a particular NetBIOS name, including the broadcast NetBIOS name ("*") defined above. A query mechanism is available to the end- node to determine if a NBDD will be able to relay a datagram to a given name. Before a datagram, or its fragments, are sent to the NBDD the P or M node may send a DATAGRAM QUERY REQUEST packet to the NBDD with the DESTINATION_NAME from the DATAGRAM SERVICE packet(s). The NBDD will respond with a DATAGRAM POSITIVE QUERY RESPONSE if it will relay datagrams to the specified destination name. After a positive response the end-node unicasts the datagram to the NBDD. If the NBDD will not be able to relay a datagram to the destination name specified in the query, a DATAGRAM NEGATIVE QUERY RESPONSE packet is returned. If the NBDD can not distribute a datagram, the end-node then has the option of getting the name's owner list from the NBNS and sending the datagram directly to each of the owners.

An NBDD must be able to respond to DATAGRAM QUERY REQUEST packets. The response may always be positive. However, the usage or implementation of the query mechanism by a P or M node is optional. An implementation may always unicast the NetBIOS datagram to the NBDD without asking if it will be relayed. Except for the datagram query facility described above, an NBDD provides no feedback to indicate whether it forwarded a datagram.

18. NODE CONFIGURATION PARAMETERS

- B NODES:
  - Node's permanent unique name
  - Whether IGMP is in use
  - Broadcast IP address to use

- Whether NetBIOS session keep-alives are needed
- Usable UDP data field length (to control fragmentation)

- P NODES:
  - Node's permanent unique name
  - IP address of NBNS
  - IP address of NBDD
  - Whether NetBIOS session keep-alives are needed
  - Usable UDP data field length (to control fragmentation)
- M NODES:
  - Node's permanent unique name
  - Whether IGMP is in use
  - Broadcast IP address to use
  - IP address of NBNS
  - IP address of NBDD
  - Whether NetBIOS session keep-alives are needed
  - Usable UDP data field length (to control fragmentation)

19. MINIMAL CONFORMANCE

To ensure multi-vendor interoperability, a minimally conforming implementation based on this specification must observe the following rules:

a) A node designed to work only in a broadcast area must conform to the B node specification.

b) A node designed to work only in an internet must conform to the P node specification.

REFERENCES

[1] "Protocol Standard For a NetBIOS Service on a TCP/UDP Transport: Detailed Specifications", RFC 1002, March 1987.

[2] IBM Corp., "IBM PC Network Technical Reference Manual", No. 6322916, First Edition, September 1984.

[3] J. Postel (Ed.), "Transmission Control Protocol", RFC 793, September 1981.

[4] MIL-STD-1778

[5] J. Postel, "User Datagram Protocol", RFC 768, 28 August 1980.

[6] J. Reynolds, J. Postel, "Assigned Numbers", RFC 990, November 1986.

[7] J. Postel, "Internet Protocol", RFC 791, September 1981.

[8] J. Mogul, "Internet Subnets", RFC 950, October 1984

[9] J. Mogul, "Broadcasting Internet Datagrams in the Presence of Subnets", RFC 922, October 1984.

[10] J. Mogul, "Broadcasting Internet Datagrams", RFC 919, October 1984.

[11] P. Mockapetris, "Domain Names - Concepts and Facilities", RFC 882, November 1983.

[12] P. Mockapetris, "Domain Names - Implementation and Specification", RFC 883, November 1983.

[13] P. Mockapetris, "Domain System Changes and Observations", RFC 973, January 1986.

[14] C. Partridge, "Mail Routing and the Domain System", RFC 974, January 1986.

[15] S. Deering, D. Cheriton, "Host Groups: A Multicast Extension to the Internet Protocol", RFC 966, December 1985.

[16] S. Deering, "Host Extensions for IP Multicasting", RFC 988, July 1986.

APPENDIX A

This appendix contains supporting technical discussions. It is not an integral part of the NetBIOS-over-TCP specification.

INTEGRATION WITH INTERNET GROUP MULTICASTING

The Netbios-over-TCP system described in this RFC may be easily integrated with the Internet Group Multicast system now being developed for the internet.

In the main body of the RFC, the notion of a broadcast area was considered to be a single MAC-bridged "B-LAN". However, the protocols defined will operate over an extended broadcast area resulting from the creation of a permanent Internet Multicast Group.

Each separate broadcast area would be based on a separate permanent Internet Multicast Group. This multicast group address would be used by B and M nodes as their BROADCAST_ADDRESS.

In order to base the broadcast area on a multicast group certain additional procedures are required and certain constraints must be met.

A-1. ADDITIONAL PROTOCOL REQUIRED IN B AND M NODES

All B or M nodes operating on an IGMP based broadcast area must have IGMP support in their IP layer software. These nodes must perform an IGMP join operation to enter the IGMP group before engaging in NetBIOS activity.

A-2. CONSTRAINTS

Broadcast Areas may overlap. For this reason, end-nodes must be careful to examine the NetBIOS scope identifiers in all received broadcast packets.

The NetBIOS broadcast protocols were designed for a network that exhibits a low average transit time and low rate of packet loss. An IGMP based broadcast area must exhibit these characteristics. In practice this will tend to constrain IGMP broadcast areas to a campus of networks interconnected by high-speed routers and inter-router links. It is unlikely that transcontinental broadcast areas would exhibit the required characteristics.

APPENDIX B

This appendix contains supporting technical discussions. It is not an integral part of the NetBIOS-over-TCP specification.

IMPLEMENTATION CONSIDERATIONS

B-1. IMPLEMENTATION MODELS

On any participating system, there must be some sort of NetBIOS Service to coordinate access by NetBIOS applications on that system.

To analyze the impact of the NetBIOS-over-TCP architecture, we use the following three models of how a NetBIOS service might be implemented:

1. Combined Service and Application Model

   The NetBIOS service and application are both contained within a single process. No interprocess communication is assumed within the system; all communication is over the network. If multiple applications require concurrent access to the NetBIOS service, they must be folded into this monolithic process.

2. Common Kernel Element Model

   The NetBIOS Service is part of the operating system (perhaps as a device driver or a front-end processor). The NetBIOS applications are normal operating system application processes. The common element NetBIOS service contains all the information, such as the name and listen tables, required to co-ordinate the activities of the applications.

3. Non-Kernel Common Element Model

   The NetBIOS Service is implemented as an operating system application process. The NetBIOS applications are other operating system application processes. The service and the applications exchange data via operating system interprocess communication. In a multi-processor (e.g. network) operating system, each module may reside on a different cpu. The NetBIOS service process contains all the shared information required to coordinate the activities of the NetBIOS applications. The applications may still require a subroutine library to facilitate access to the NetBIOS service.

For any of the implementation models, the TCP/IP service can be located in the operating system or split among the NetBIOS applications and the NetBIOS service processes.

B-1.1 MODEL INDEPENDENT CONSIDERATIONS

The NetBIOS name service associates a NetBIOS name with a host. The NetBIOS session service further binds the name to a specific TCP port for the duration of the session.

The name service does not need to be informed of every Listen initiation and completion. Since the names are not bound to any TCP port in the name service, the session service may use a different tcp port for each session established with the same local name.

The TCP port used for the data transfer phase of a NetBIOS session can be globally well-known, locally well-known, or ephemeral. The choice is a local implementation issue. The RETARGET mechanism allows the binding of the NetBIOS session to a TCP connection to any TCP port, even to another IP node.

An implementation may use the session service's globally well- known TCP port for the data transfer phase of the session by not using the RETARGET mechanism and, rather, accepting the session on the initial TCP connection. This is permissible because the caller always uses an ephemeral TCP port.

The complexity of the called end RETARGET mechanism is only required if a particular implementation needs it. For many real system environments, such as an in-kernel NetBIOS service implementation, it will not be necessary to retarget incoming calls. Rather, all NetBIOS sessions may be multiplexed through the single, well-known, NetBIOS session service port. These implementations will not be burdened by the complexity of the RETARGET mechanism, nor will their callers be required to jump through the retargetting hoops.

Nevertheless, all callers must be ready to process all possible SESSION RETARGET RESPONSEs.

B-1.2 SERVICE OPERATION FOR EACH MODEL

It is possible to construct a NetBIOS service based on this specification for each of the above defined implementation models.

For the common kernel element model, all the NetBIOS services, name, datagram, and session, are simple. All the information is contained within a single entity and can therefore be accessed or modified easily. A single port or multiple ports for the NetBIOS sessions can be used without adding any significant complexity to the session establishment procedure. The only penalty is the amount of overhead incurred to get the NetBIOS messages and operation requests/responses

through the user and operating system boundary.

The combined service and application model is very similar to the common kernel element model in terms of its requirements on the NetBIOS service. The major difficulty is the internal coordination of the multiple NetBIOS service and application processes existing in a system of this type.

The NetBIOS name, datagram and session protocols assume that the entities at the end-points have full control of the various well-known TCP and UDP ports. If an implementation has multiple NetBIOS service entities, as would be the case with, for example, multiple applications each linked into a NetBIOS library, then that implementation must impose some internal coordination. Alternatively, use of the NetBIOS ports could be periodically assigned to one application or another.

For the typical non-kernel common element mode implementation, three permanent system-wide NetBIOS service processes would exist:

- The name server
- the datagram server
- and session server

Each server would listen for requests from the network on a UDP or TCP well-known port. Each application would have a small piece of the NetBIOS service built-in, possibly a library. Each application's NetBIOS support library would need to send a message to the particular server to request an operation, such as add name or send a datagram or set-up a listen.

The non-kernel common element model does not require a TCP connection be passed between the two processes, session server and application. The RETARGET operation for an active NetBIOS Listen could be used by the session server to redirect the session to another TCP connection on a port allocated and owned by the application's NetBIOS support library. The application with either a built-in or a kernel-based TCP/IP service could then accept the RETARGETed connection request and process it independently of the session server.

On Unix(tm) or POSIX(tm), the NetBIOS session server could create sub-processes for incoming connections. The open sessions would be passed through "fork" and "exec" to the child as an open file descriptor. This approach is very limited, however. A pre- existing process could not receive an incoming call. And all call-ed processes would have to be sub-processes of the session server.

B-2. CASUAL AND RESTRICTED NetBIOS APPLICATIONS

Because NetBIOS was designed to operate in the open system environment of the typical personal computer, it does not have the

concept of privileged or unprivileged applications. In many multi-user or multi-tasking operating systems applications are assigned privilege capabilities. These capabilities limit the applications ability to acquire and use system resources. For these systems it is important to allow casual applications, those with limited system privileges, and privileged applications, those with 'super-user' capabilities but access to them and their required resources is restricted, to access NetBIOS services. It is also important to allow a systems administrator to restrict certain NetBIOS resources to a particular NetBIOS application. For example, a file server based on the NetBIOS services should be able to have names and TCP ports for sessions only it can use.

A NetBIOS application needs at least two local resources to communicate with another NetBIOS application, a NetBIOS name for itself and, typically, a session. A NetBIOS service cannot require that NetBIOS applications directly use privileged system resources. For example, many systems require privilege to use TCP and UDP ports with numbers less than 1024. This RFC requires reserved ports for the name and session servers of a NetBIOS service implementation. It does not require the application to have direct access these reserved ports.

For the name service, the manager of the local name table must have access to the NetBIOS name service's reserved UDP port. It needs to listen for name service UDP packets to defend and define its local names to the network. However, this manager need not be a part of a user application in a system environment which has privilege restrictions on reserved ports.

The internal name server can require certain privileges to add, delete, or use a certain name, if an implementer wants the restriction. This restriction is independent of the operation of the NetBIOS service protocols and would not necessarily prevent the interoperation of that implementation with another implementation.

The session server is required to own a reserved TCP port for session establishment. However, the ultimate TCP connection used to transmit and receive data does not have to be through that reserved port. The RETARGET procedure the NetBIOS session to be shifted to another TCP connection, possibly through a different port at the called end. This port can be an unprivileged resource, with a value greater than 1023. This facilitates casual applications.

Alternately, the RETARGET mechanism allows the TCP port used for data transmission and reception to be a reserved port. Consequently, an application wishing to have access to its ports maintained by the system administrator can use these restricted TCP ports. This facilitates privileged applications.

A particular implementation may wish to require further special

privileges for session establishment, these could be associated with internal information. It does not have to be based solely on TCP port allocation. For example, a given NetBIOS name may only be used for sessions by applications with a certain system privilege level.

The decision to use reserved or unreserved ports or add any additional name registration and usage authorization is a purely local implementation decision. It is not required by the NetBIOS protocols specified in the RFC.

B-3. TCP VERSUS SESSION KEEP-ALIVES

The KEEP-ALIVE is a protocol element used to validate the existence of a connection. A packet is sent to the remote connection partner to solicit a response which shows the connection is still functioning. TCP KEEP-ALIVES are used at the TCP level on TCP connections while session KEEP-ALIVES are used on NetBIOS sessions. These protocol operations are always transparent to the connection user. The user will only find out about a KEEP-ALIVE operation if it fails, therefore, if the connection is lost.

The NetBIOS specification[2] requires the NetBIOS service to inform the session user if a session is lost when it is in a passive or active state. Therefore,if the session user is only waiting for a receive operation and the session is dropped the NetBIOS service must inform the session user. It cannot wait for a session send operation before it informs the user of the loss of the connection.

This requirement stems from the management of scarce or volatile resources by a NetBIOS application. If a particular user terminates a session with a server application by destroying the client application or the NetBIOS service without a NetBIOS Hang Up, the server application will want to clean-up or free allocated resources. This server application if it only receives and then sends a response requires the notification of the session abort in the passive state.

The standard definition of a TCP service cannot detect loss of a connection when it is in a passive state, waiting for a packet to arrive. Some TCP implementations have added a KEEP-ALIVE operation which is interoperable with implementations without this feature. These implementations send a packet with an invalid sequence number to the connection partner. The partner, by specification, must respond with a packet showing the correct sequence number of the connection. If no response is received from the remote partner within a certain time interval then the TCP service assumes the connection is lost.

Since many TCP implementations do not have this KEEP-ALIVE function an optional NetBIOS KEEP-ALIVE operation has been added to the NetBIOS session protocols. The NetBIOS KEEP-ALIVE uses the properties of TCP to solicit a response from the remote connection

partner. A NetBIOS session message called KEEP-ALIVE is sent to the remote partner. Since this results in TCP sending an IP packet to the remote partner, the TCP connection is active. TCP will discover if the TCP connection is lost if the remote TCP partner does not acknowledge the IP packet. Therefore, the NetBIOS session service does not send a response to a session KEEP ALIVE message. It just throws it away. The NetBIOS session service that transmits the KEEP ALIVE is informed only of the failure of the TCP connection. It does not wait for a specific response message.

A particular NetBIOS implementation should use KEEP-ALIVES if it is concerned with maintaining compatibility with the NetBIOS interface specification[2]. Compatibility is especially important if the implementation wishes to support existing NetBIOS applications, which typically require the session loss detection on their servers, or future applications which were developed for implementations with session loss detection.

B-4. RETARGET ALGORITHMS

This section contains 2 suggestions for RETARGET algorithms. They are called the "straight" and "stack" methods. The algorithm in the body of the RFC uses the straight method. Implementation of either algorithm must take into account the Session establishment maximum retry count. The retry count is the maximum number of TCP connect operations allowed before a failure is reported.

The straight method forces the session establishment procedure to begin a retry after a retargetting failure with the initial node returned from the name discovery procedure. A retargetting failure is when a TCP connection attempt fails because of a time- out or a NEGATIVE SESSION RESPONSE is received with an error code specifying NOT LISTENING ON CALLED NAME. If any other failure occurs the session establishment procedure should retry from the call to the name discovery procedure.

A minimum of 2 retries, either from a retargetting or a name discovery failure. This will give the session service a chance to re-establish a NetBIOS Listen or, more importantly, allow the NetBIOS scope, local name service or the NBNS, to re-learn the correct IP address of a NetBIOS name.

The stack method operates similarly to the straight method. However, instead of retrying at the initial node returned by the name discovery procedure, it restarts with the IP address of the last node which sent a SESSION RETARGET RESPONSE prior to the retargetting failure. To limit the stack method, any one host can only be tried a maximum of 2 times.

B-5. NBDD SERVICE

If the NBDD does not forward datagrams then don't provide Group and Broadcast NetBIOS datagram services to the NetBIOS user. Therefore, ignore the implementation of the query request and, when get a negative response, acquiring the membership list of IP addresses and sending the datagram as a unicast to each member.

B-6. APPLICATION CONSIDERATIONS

B-6.1 USE OF NetBIOS DATAGRAMS

Certain existing NetBIOS applications use NetBIOS datagrams as a foundation for their own connection-oriented protocols. This can cause excessive NetBIOS name query activity and place a substantial burden on the network, server nodes, and other end- nodes. It is recommended that this practice be avoided in new applications.

Appendix C

RFC 1002

This appendix reproduces, in full and unedited, RFC 1002, Protocol Standard for a NetBIOS Service on a TCP/UDP Transport: Detailed Specifications.

Network Working Group
Request for Comments: 1002 March, 1987

PROTOCOL STANDARD FOR A NetBIOS SERVICE ON A TCP/UDP TRANSPORT: DETAILED SPECIFICATIONS

ABSTRACT

This RFC defines a proposed standard protocol to support NetBIOS services in a TCP/IP environment. Both local network and internet operation are supported. Various node types are defined to accommodate local and internet topologies and to allow operation with or without the use of IP broadcast.

This RFC gives the detailed specifications of the NetBIOS-over-TCP packets, protocols, and defined constants and variables. A more general overview is found in a companion RFC, "Protocol Standard For a NetBIOS Service on a TCP/UDP Transport: Concepts and Methods".

TABLE OF CONTENTS

PROTOCOL STANDARD FOR A NetBIOS SERVICE ON A TCP/UDP TRANSPORT: DETAILED SPECIFICATIONS

1. STATUS OF THIS MEMO

This RFC specifies a proposed standard for the DARPA Internet community. Since this topic is new to the Internet community, discussions and suggestions are specifically requested.

Please send written comments to:

Karl Auerbach
Epilogue Technology Corporation
P.O. Box 5432
Redwood City, CA 94063

Please send online comments to:

Avnish Aggarwal
Internet: mtxinu!excelan!avnish@ucbvax.berkeley.edu
Usenet: ucbvax!mtxinu!excelan!avnish

Distribution of this memorandum is unlimited.

2. ACKNOWLEDGEMENTS

This RFC has been developed under the auspices of the Internet Activities Board.

The following individuals have contributed to the development of this RFC:

Avnish Aggarwal	Arvind Agrawal	Lorenzo Aguilar
Geoffrey Arnold	Karl Auerbach	K. Ramesh Babu
Keith Ball	Amatzia Ben-Artzi	Vint Cerf
Richard Cherry	David Crocker	Steve Deering
Greg Ennis	Steve Holmgren	Jay Israel
David Kaufman	Lee LaBarre	James Lau
Dan Lynch	Gaylord Miyata	David Stevens
Steve Thomas	Ishan Wu	

The system proposed by this RFC does not reflect any existing Netbios-over-TCP implementation. However, the design incorporates considerable knowledge obtained from prior implementations. Special thanks goes to the following organizations which have provided this invaluable information:

CMC/Syros Excelan Sytek Ungermann-Bass

3. INTRODUCTION

This RFC contains the detailed packet formats and protocol specifications for NetBIOS-over-TCP. This RFC is a companion to RFC 1001, "Protocol Standard For a NetBIOS Service on a TCP/UDP Transport: Concepts and Methods" [1].

4. PACKET DESCRIPTIONS

Bit and byte ordering are defined by the most recent version of "Assigned Numbers" [2].

4.1. NAME FORMAT

The NetBIOS name representation in all NetBIOS packets (for NAME, SESSION, and DATAGRAM services) is defined in the Domain Name Service RFC 883[3] as "compressed" name messages. This format is called "second-level encoding" in the section entitled "Representation of NetBIOS Names" in the Concepts and Methods document.

For ease of description, the first two paragraphs from page 31, the section titled "Domain name representation and compression", of RFC 883 are replicated here:

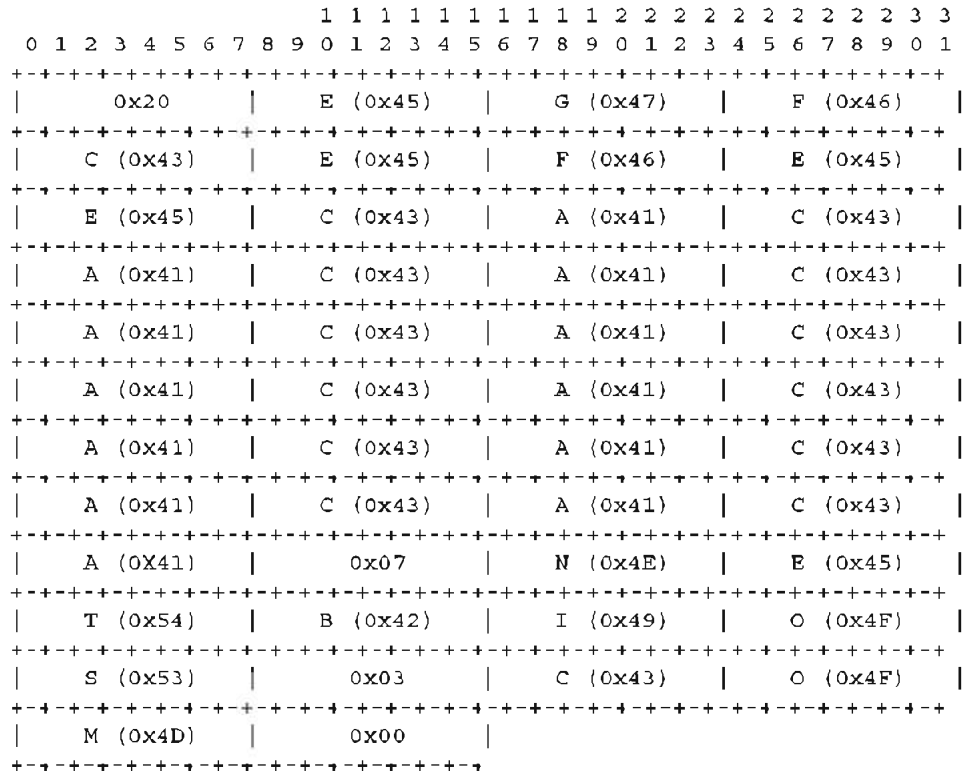
> Domain names messages are expressed in terms of a sequence of labels. Each label is represented as a one octet length field followed by that number of octets. Since every domain name ends with the null label of the root, a compressed domain name is terminated by a length byte of zero. The high order two bits of the length field must be zero, and the remaining six bits of the length field limit the label to 63 octets or less.
>
> To simplify implementations, the total length of label octets and label length octets that make up a domain name is restricted to 255 octets or less.

The following is the uncompressed representation of the NetBIOS name "FRED ", which is the 4 ASCII characters, F, R, E, D, followed by 12 space characters (0x20). This name has the SCOPE_ID: "NETBIOS.COM"

EGFCEFEECACACACACACACACACACACACA.NETBIOS.COM

This uncompressed representation of names is called "first-level encoding" in the section entitled "Representation of NetBIOS Names" in the Concepts and Methods document.

The following is a pictographic representation of the compressed representation of the previous uncompressed Domain Name representation.



Each section of a domain name is called a label [7 (page 31)]. A label can be a maximum of 63 bytes. The first byte of a label in compressed representation is the number of bytes in the label. For the above example, the first 0x20 is the number of bytes in the left-most label, EGFCEFEECACACACACACACACACACACA, of the domain name. The bytes following the label length count are the characters of the label. The following labels are in sequence after the first label, which is the encoded NetBIOS name, until a zero (0x00) length count. The zero length count represents the root label, which is always null.

A label length count is actually a 6-bit field in the label length field. The most significant 2 bits of the field, bits 7 and 6, are flags allowing an escape from the above compressed representation. If bits 7 and 6 are both set (11), the following 14 bits are an offset pointer into the full message to the actual label string from another domain name that belongs in this name. This label pointer allows for a further compression of a domain name in a packet.

NetBIOS implementations can only use label string pointers in Name Service packets. They cannot be used in Session or Datagram Service packets.

The other two possible values for bits 7 and 6 (01 and 10) of a label length field are reserved for future use by RFC 883[2 (page 32)].

Note that the first octet of a compressed name must contain one of the following bit patterns. (An "x" indicates a bit whose value may be either 0 or 1.):

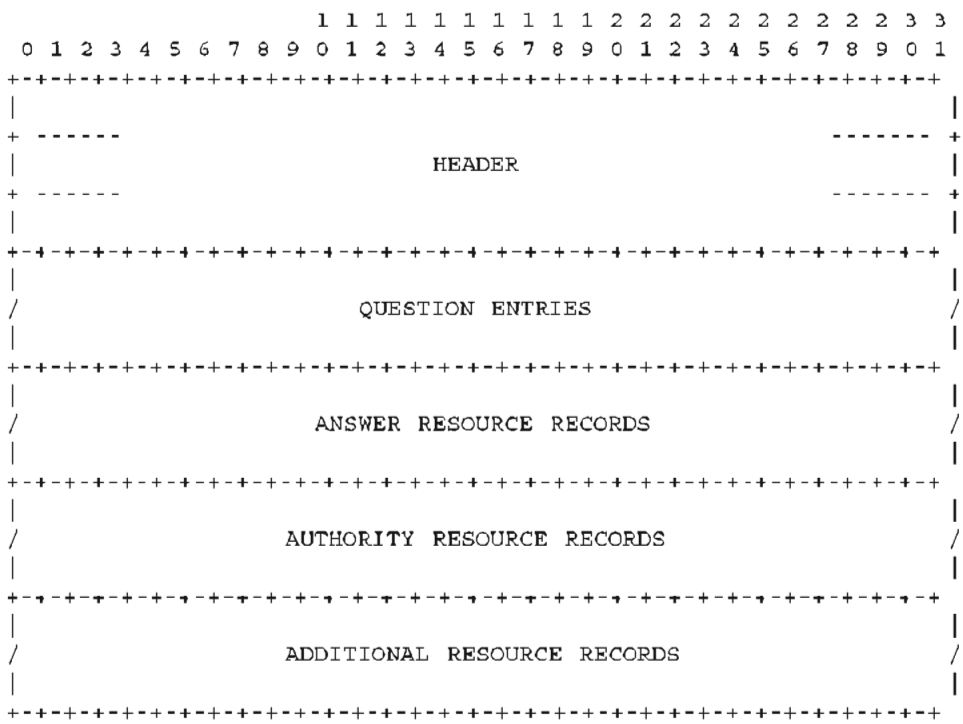
```
00100000 -  Netbios name, length must be 32 (decimal)
11xxxxxx -  Label string pointer
10xxxxxx -  Reserved
01xxxxxx -  Reserved
```

4.2. NAME SERVICE PACKETS

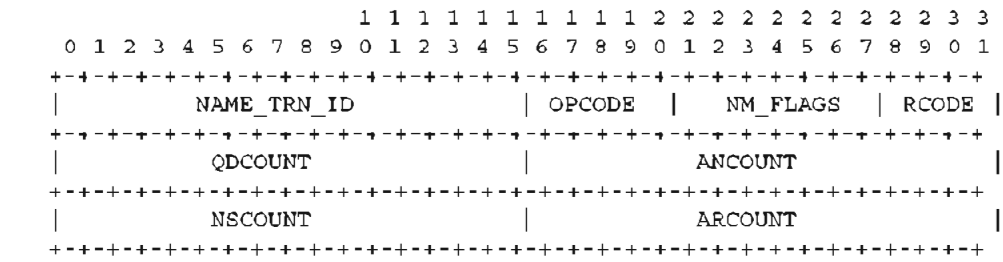
4.2.1. GENERAL FORMAT OF NAME SERVICE PACKETS

The NetBIOS Name Service packets follow the packet structure defined in the Domain Name Service (DNS) RFC 883 [7 (pg 26-31)]. The structures are compatible with the existing DNS packet formats, however, additional types and codes have been added to work with NetBIOS.

If Name Service packets are sent over a TCP connection they are preceded by a 16 bit unsigned integer representing the length of the Name Service packet.

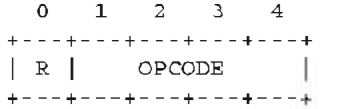


4.2.1.1. HEADER



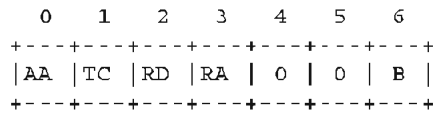
Field	Description
NAME_TRN_ID	Transaction ID for Name Service Transaction. Requestor places a unique value for each active transaction. Responder puts NAME_TRN_ID value from request packet in response packet.
OPCODE	Packet type code, see table below.
NM_FLAGS	Flags for operation, see table below.
RCODE	Result codes of request. Table of RCODE values for each response packet below.
QDCOUNT	Unsigned 16 bit integer specifying the number of entries in the question section of a Name Service packet. Always zero (0) for responses. Must be non-zero for all NetBIOS Name requests.
ANCOUNT	Unsigned 16 bit integer specifying the number of resource records in the answer section of a Name Service packet.
NSCOUNT	Unsigned 16 bit integer specifying the number of resource records in the authority section of a Name Service packet.
ARCOUNT	Unsigned 16 bit integer specifying the number of resource records in the additional records section of a Name Service packet.

The OPCODE field is defined as:



Symbol	Bit(s)	Description
OPCODE	1-4	Operation specifier: 0 = query 5 = registration 6 = release 7 = WACK 8 = refresh
R	0	RESPONSE flag: if bit == 0 then request packet if bit == 1 then response packet.

The NM_FLAGS field is defined as:



Symbol	Bit(s)	Description
B	6	Broadcast Flag. = 1: packet was broadcast or multicast = 0: unicast
RA	3	Recursion Available Flag. Only valid in responses from a NetBIOS Name Server -- must be zero in all other responses. If one (1) then the NBNS supports recursive query, registration, and release. If zero (0) then the end-node must iterate for query and challenge for registration.
RD	2	Recursion Desired Flag. May only be set on a request to a NetBIOS Name Server. The NBNS will copy its state into the response packet. If one (1) the NBNS will iterate on the query, registration, or release.
TC	1	Truncation Flag.

Set if this message was truncated because the datagram carrying it would be greater than 576 bytes in length. Use TCP to get the information from the NetBIOS Name Server.

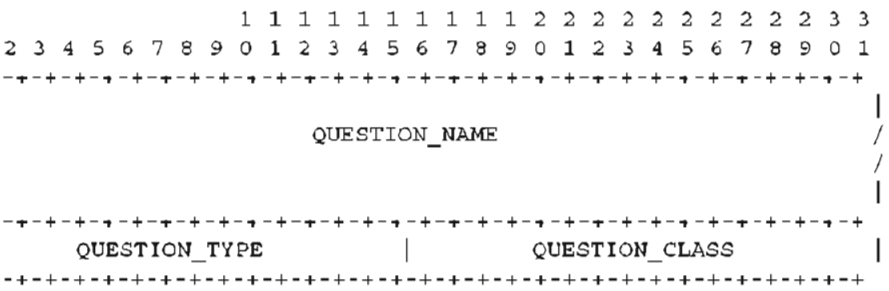
AA 0 Authoritative Answer flag.

Must be zero (0) if R flag of OPCODE is zero (0).

If R flag is one (1) then if AA is one (1) then the node responding is an authority for the domain name.

End nodes responding to queries always set this bit in responses.

4.2.1.2. QUESTION SECTION



Field	Description
QUESTION_NAME	The compressed name representation of the NetBIOS name for the request.
QUESTION_TYPE	The type of request. The values for this field are specified for each request.
QUESTION_CLASS	The class of the request. The values for this field are specified for each request.

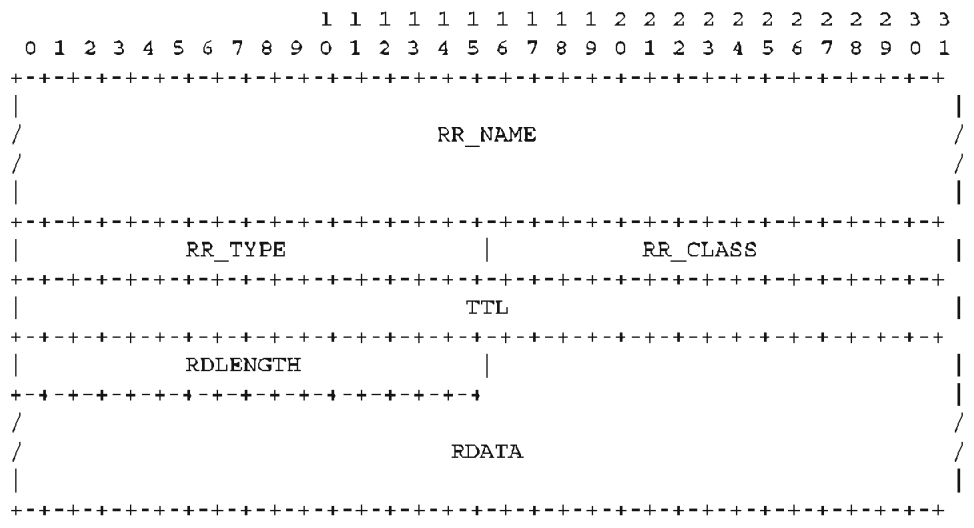
QUESTION_TYPE is defined as:

Symbol	Value	Description:
NB	0x0020	NetBIOS general Name Service Resource Record
NBSTAT	0x0021	NetBIOS NODE STATUS Resource Record (See NODE STATUS REQUEST)

QUESTION_CLASS is defined as:

Symbol	Value	Description:
IN	0x0001	Internet class

4.2.1.3. RESOURCE RECORD



Field	Description
RR_NAME	The compressed name representation of the NetBIOS name corresponding to this resource record.
RR_TYPE	Resource record type code
RR_CLASS	Resource record class code
TTL	The Time To Live of a the resource record's name.
RDLENGTH	Unsigned 16 bit integer that specifies the number of bytes in the RDATA field.
RDATA	RR_CLASS and RR_TYPE dependent field. Contains the resource information for the NetBIOS name.

RESOURCE RECORD RR_TYPE field definitions:

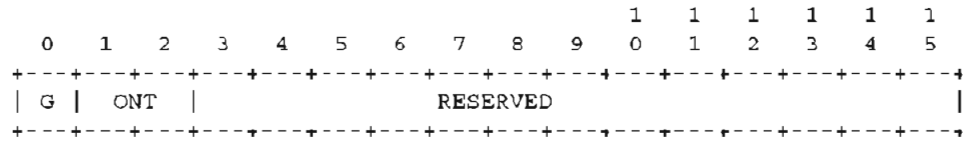
Symbol	Value	Description:
A	0x0001	IP address Resource Record (See REDIRECT NAME QUERY RESPONSE)
NS	0x0002	Name Server Resource Record (See REDIRECT

		NAME QUERY RESPONSE)
NULL	0x000A	NULL Resource Record (See WAIT FOR ACKNOWLEDGEMENT RESPONSE)
NB	0x0020	NetBIOS general Name Service Resource Record (See NB_FLAGS and NB_ADDRESS, below)
NBSTAT	0x0021	NetBIOS NODE STATUS Resource Record (See NODE STATUS RESPONSE)

RESOURCE RECORD RR_CLASS field definitions:

Symbol	Value	Description:
IN	0x0001	Internet class

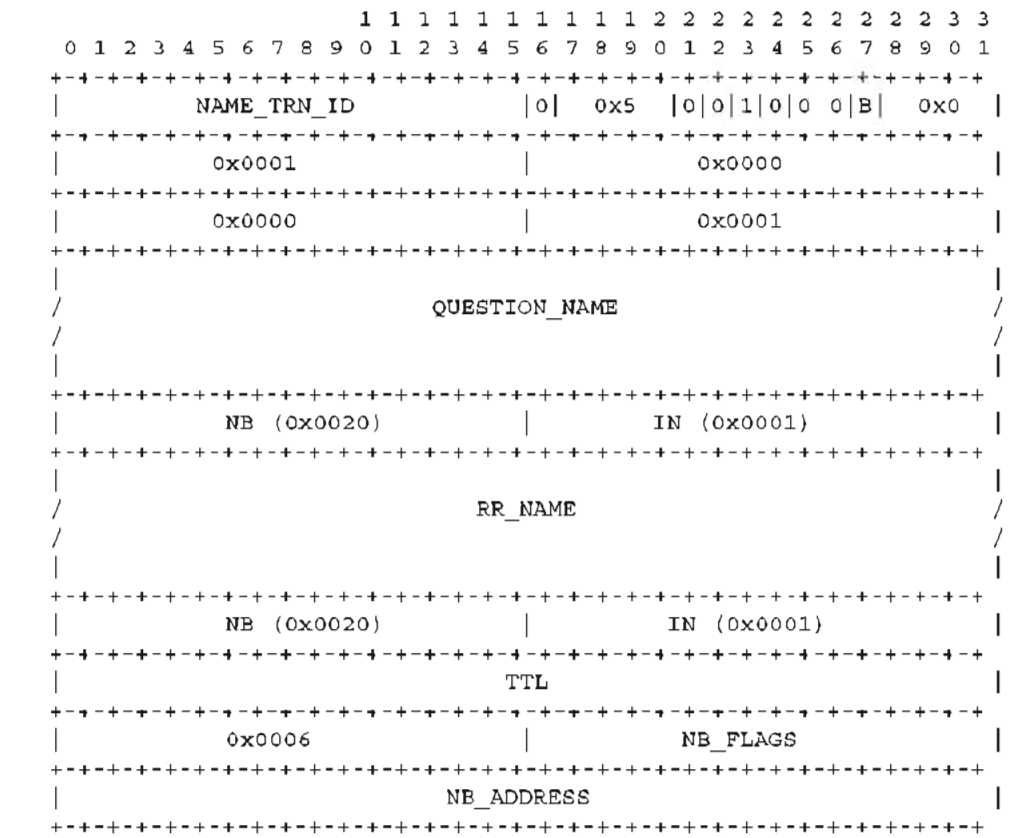
NB_FLAGS field of the RESOURCE RECORD RDATA field for RR_TYPE of "NB":



Symbol	Bit(s)	Description:
RESERVED	3-15	Reserved for future use. Must be zero (0).
ONT	1,2	Owner Node Type: 00 = B node 01 = P node 10 = M node 11 = Reserved for future use For registration requests this is the claimant's type. For responses this is the actual owner's type.
G	0	Group Name Flag. If one (1) then the RR_NAME is a GROUP NetBIOS name. If zero (0) then the RR_NAME is a UNIQUE NetBIOS name.

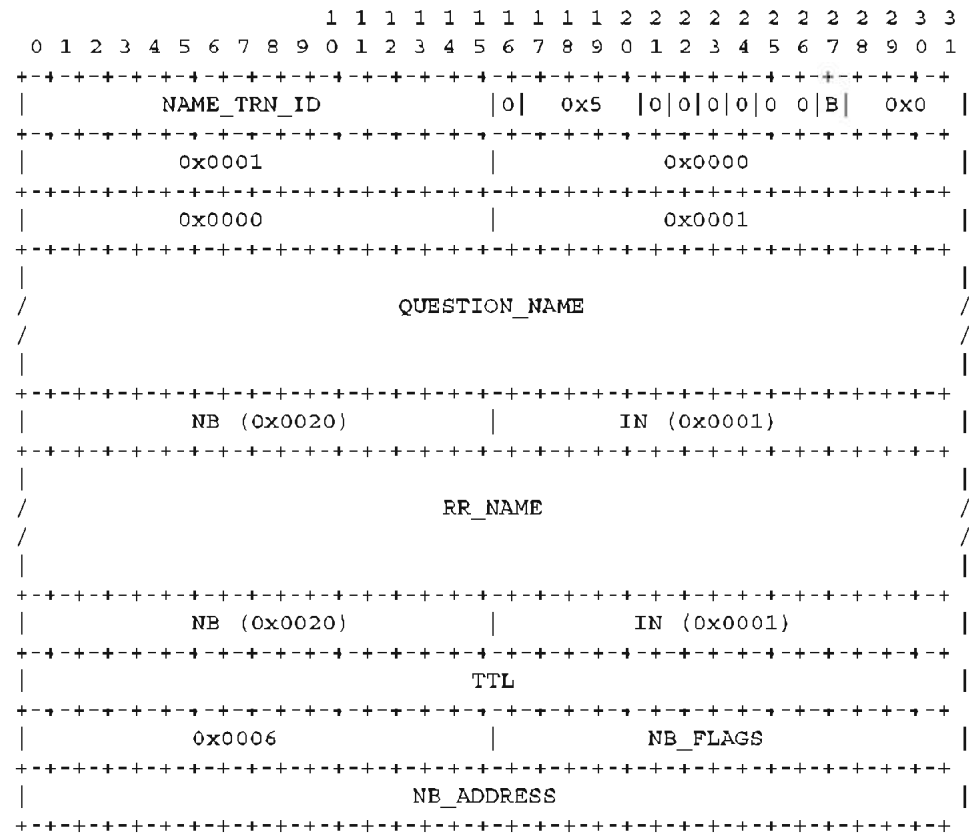
The NB_ADDRESS field of the RESOURCE RECORD RDATA field for RR_TYPE of "NB" is the IP address of the name's owner.

4.2.2. NAME REGISTRATION REQUEST

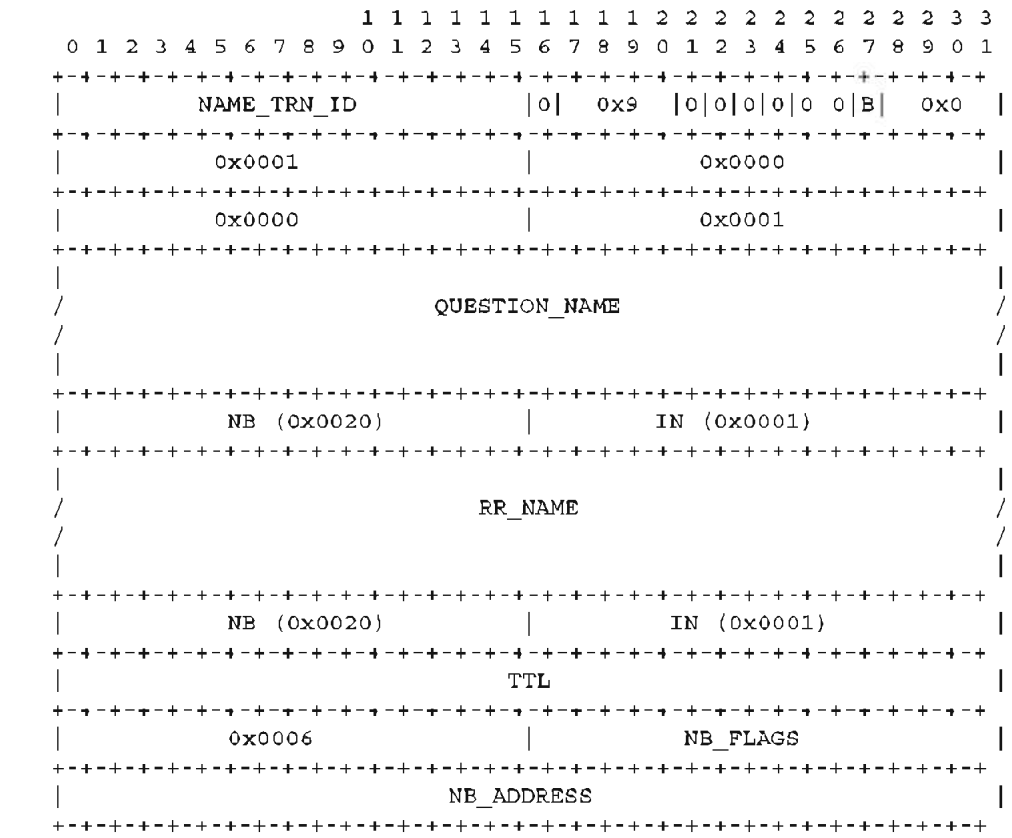


Since the RR_NAME is the same name as the QUESTION_NAME, the RR_NAME representation must use pointers to the QUESTION_NAME name's labels to guarantee the length of the datagram is less than the maximum 576 bytes. See section above on name formats and also page 31 and 32 of RFC 883, Domain Names - Implementation and Specification, for a complete description of compressed name label pointers.

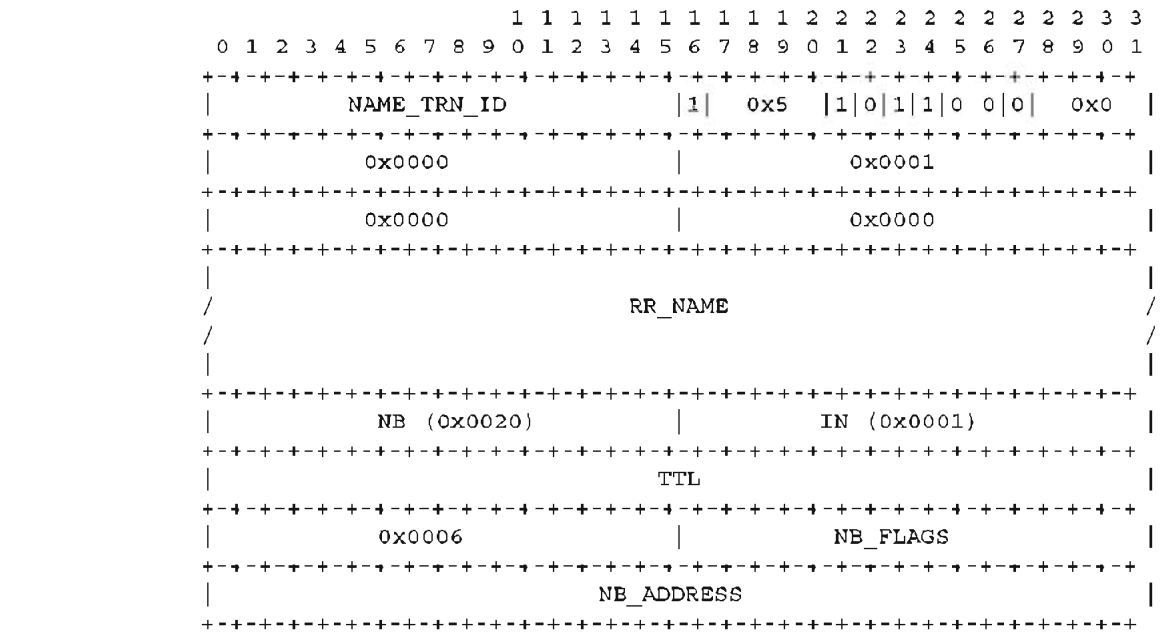
4.2.3. NAME OVERWRITE REQUEST & DEMAND



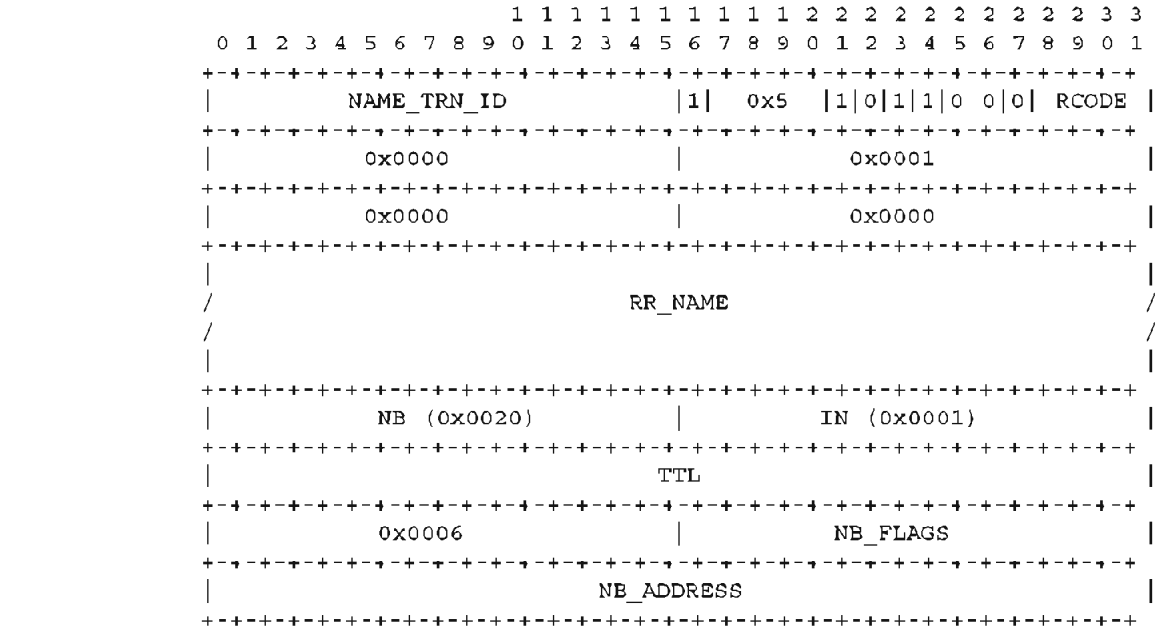
4.2.4. NAME REFRESH REQUEST



4.2.5. POSITIVE NAME REGISTRATION RESPONSE



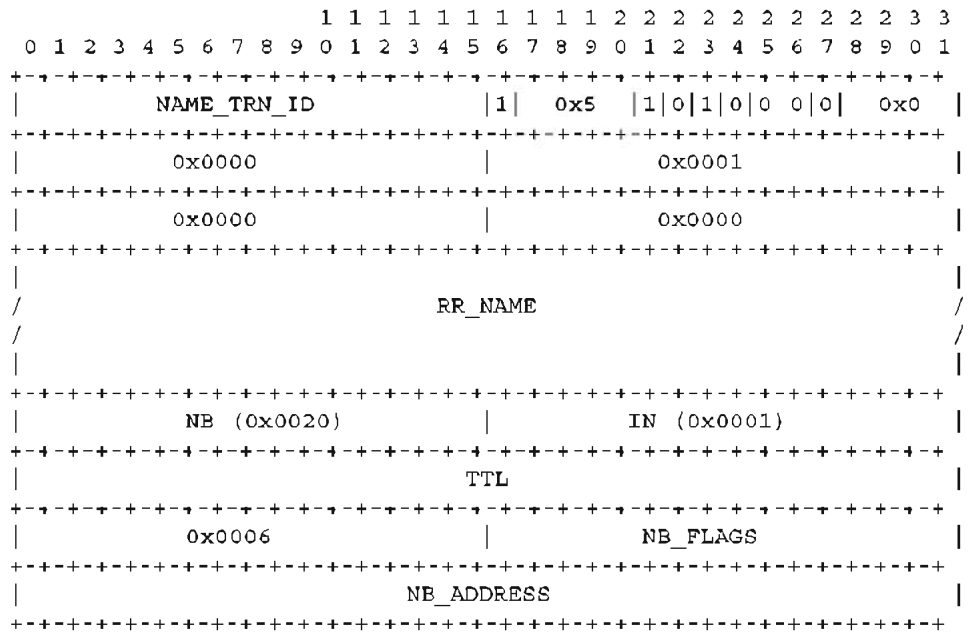
4.2.6. NEGATIVE NAME REGISTRATION RESPONSE



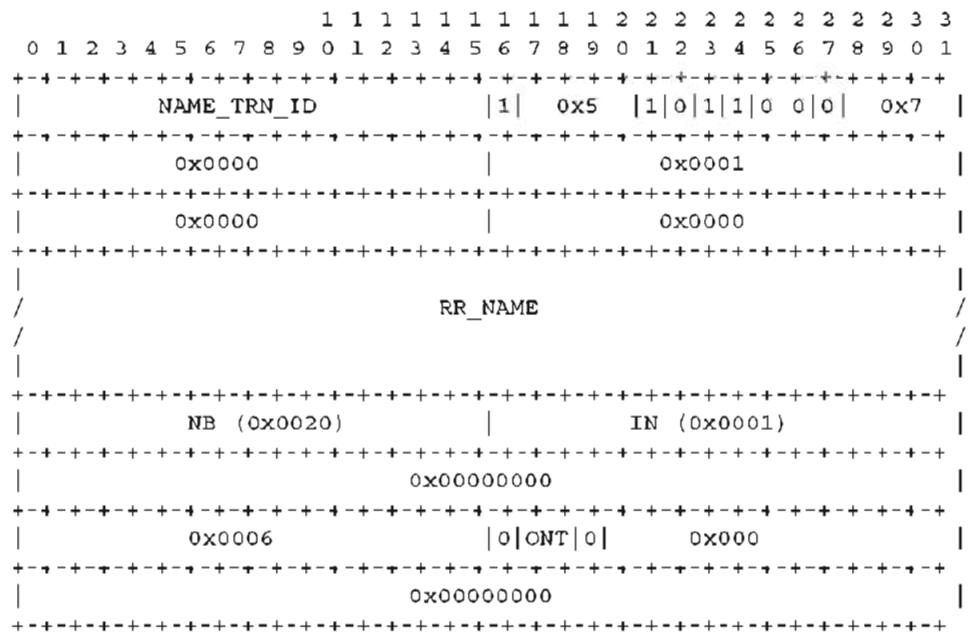
RCODE field values:

Symbol	Value	Description:
FMT_ERR	0x1	Format Error. Request was invalidly formatted.
SRV_ERR	0x2	Server failure. Problem with NBNS, cannot process name.
IMP_ERR	0x4	Unsupported request error. Allowable only for challenging NBNS when gets an Update type registration request.
RFS_ERR	0x5	Refused error. For policy reasons server will not register this name from this host.
ACT_ERR	0x6	Active error. Name is owned by another node.
CFT_ERR	0x7	Name in conflict error. A UNIQUE name is owned by more than one node.

4.2.7. END-NODE CHALLENGE REGISTRATION RESPONSE

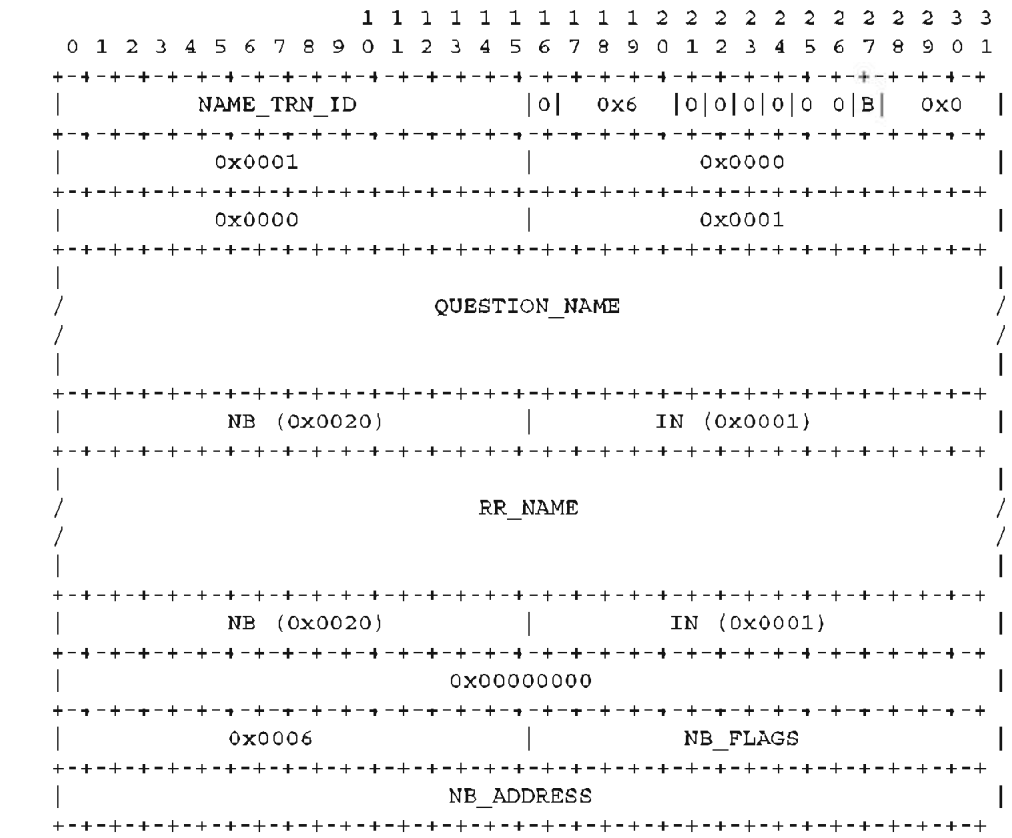


4.2.8. NAME CONFLICT DEMAND



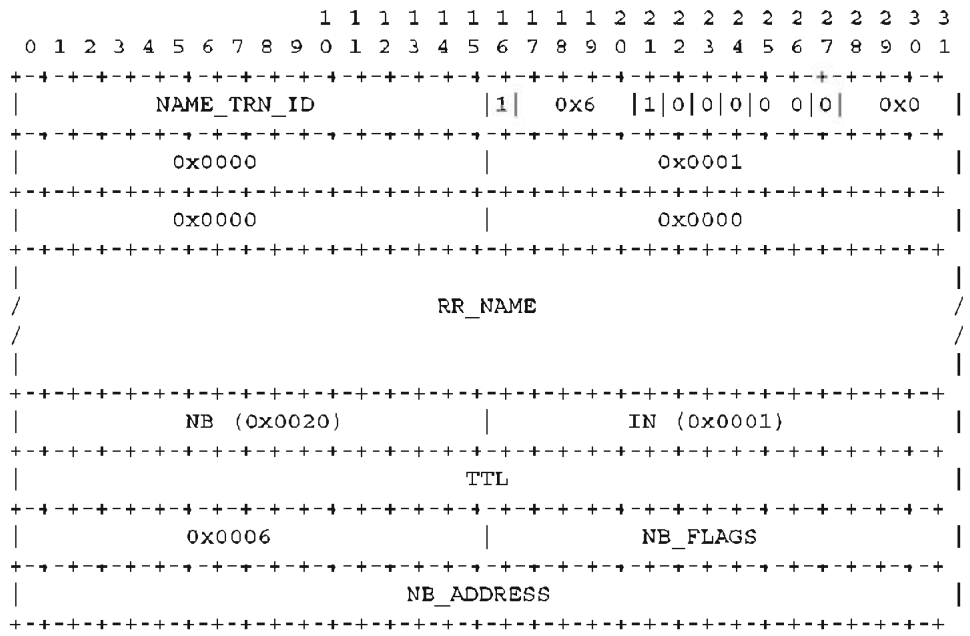
This packet is identical to a NEGATIVE NAME REGISTRATION RESPONSE with RCODE = CFT_ERR.

4.2.9. NAME RELEASE REQUEST & DEMAND

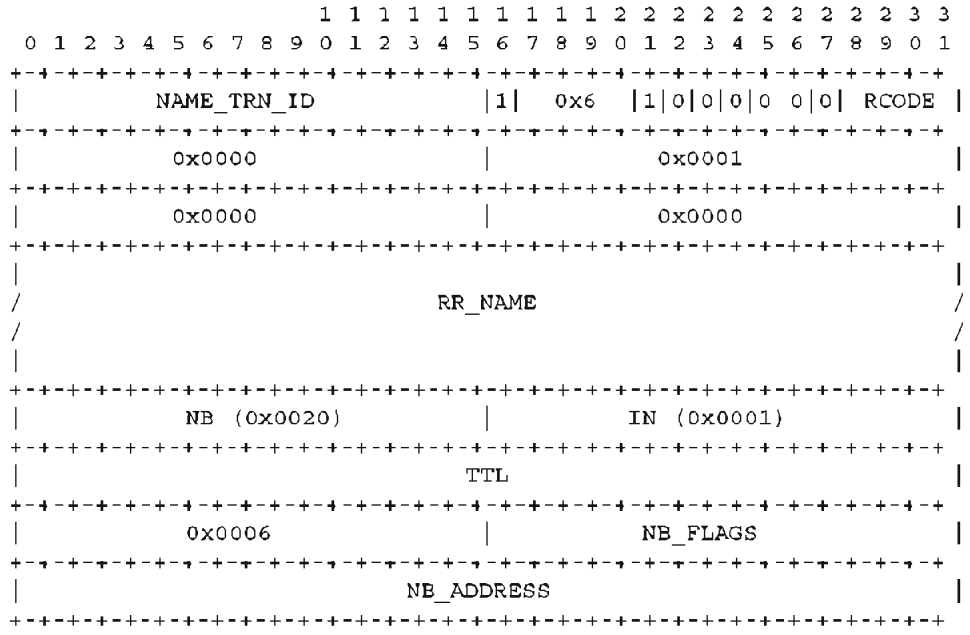


Since the RR_NAME is the same name as the QUESTION_NAME, the RR_NAME representation must use label string pointers to the QUESTION_NAME labels to guarantee the length of the datagram is less than the maximum 576 bytes. This is the same condition as with the NAME REGISTRATION REQUEST.

4.2.10. POSITIVE NAME RELEASE RESPONSE



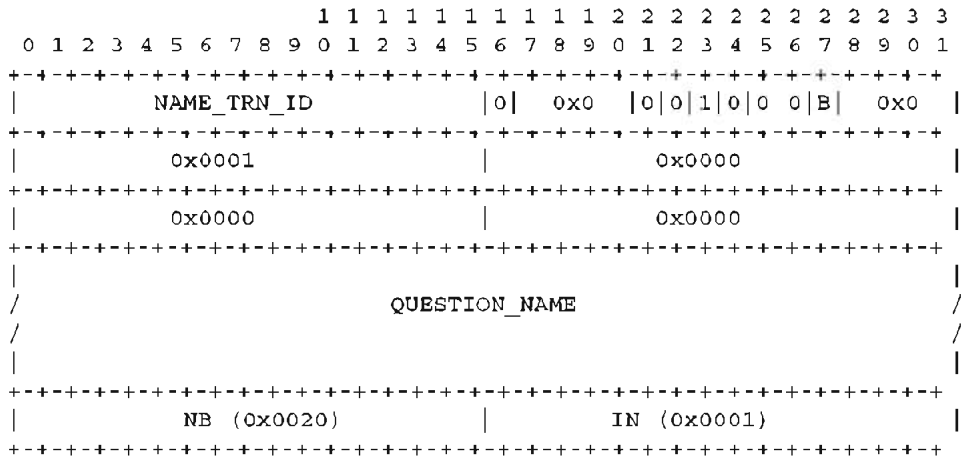
4.2.11. NEGATIVE NAME RELEASE RESPONSE



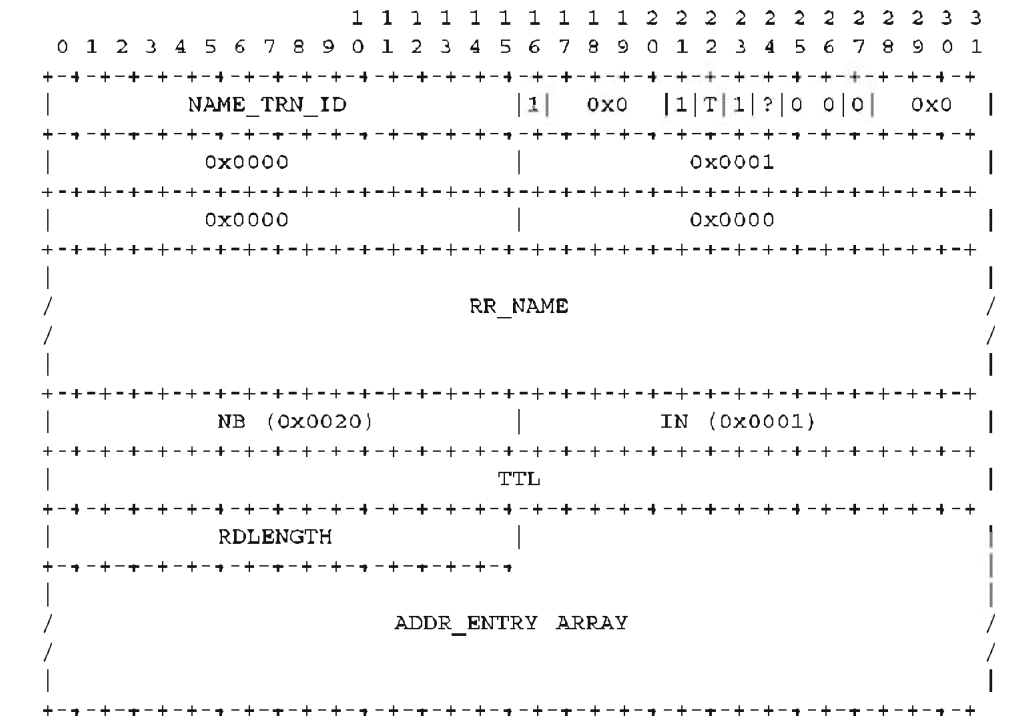
RCODE field values:

Symbol	Value	Description:
FMT_ERR	0x1	Format Error. Request was invalidly formatted.
SRV_ERR	0x2	Server failure. Problem with NBNS, cannot process name.
RFS_ERR	0x5	Refused error. For policy reasons server will not release this name from this host.
ACT_ERR	0x6	Active error. Name is owned by another node. Only that node may release it. A NetBIOS Name Server can optionally allow a node to release a name it does not own. This would facilitate detection of inactive names for nodes that went down silently.

4.2.12. NAME QUERY REQUEST

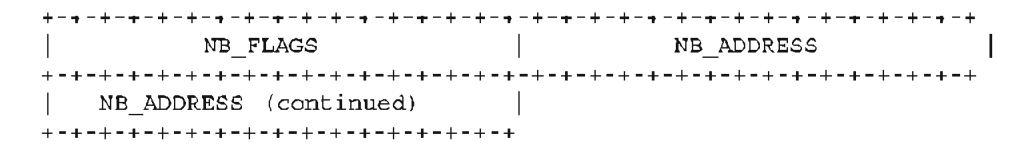


4.2.13. POSITIVE NAME QUERY RESPONSE

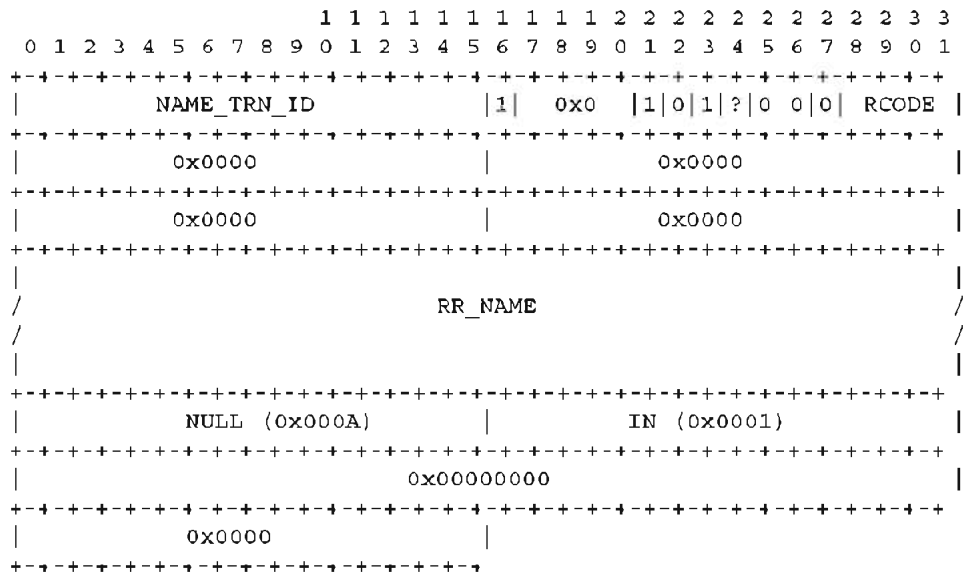


The ADDR_ENTRY ARRAY a sequence of zero or more ADDR_ENTRY records. Each ADDR_ENTRY record represents an owner of a name. For group names there may be multiple entries. However, the list may be incomplete due to packet size limitations. Bit 22, "T", will be set to indicate truncated data.

Each ADDR_ENTRY has the following format:



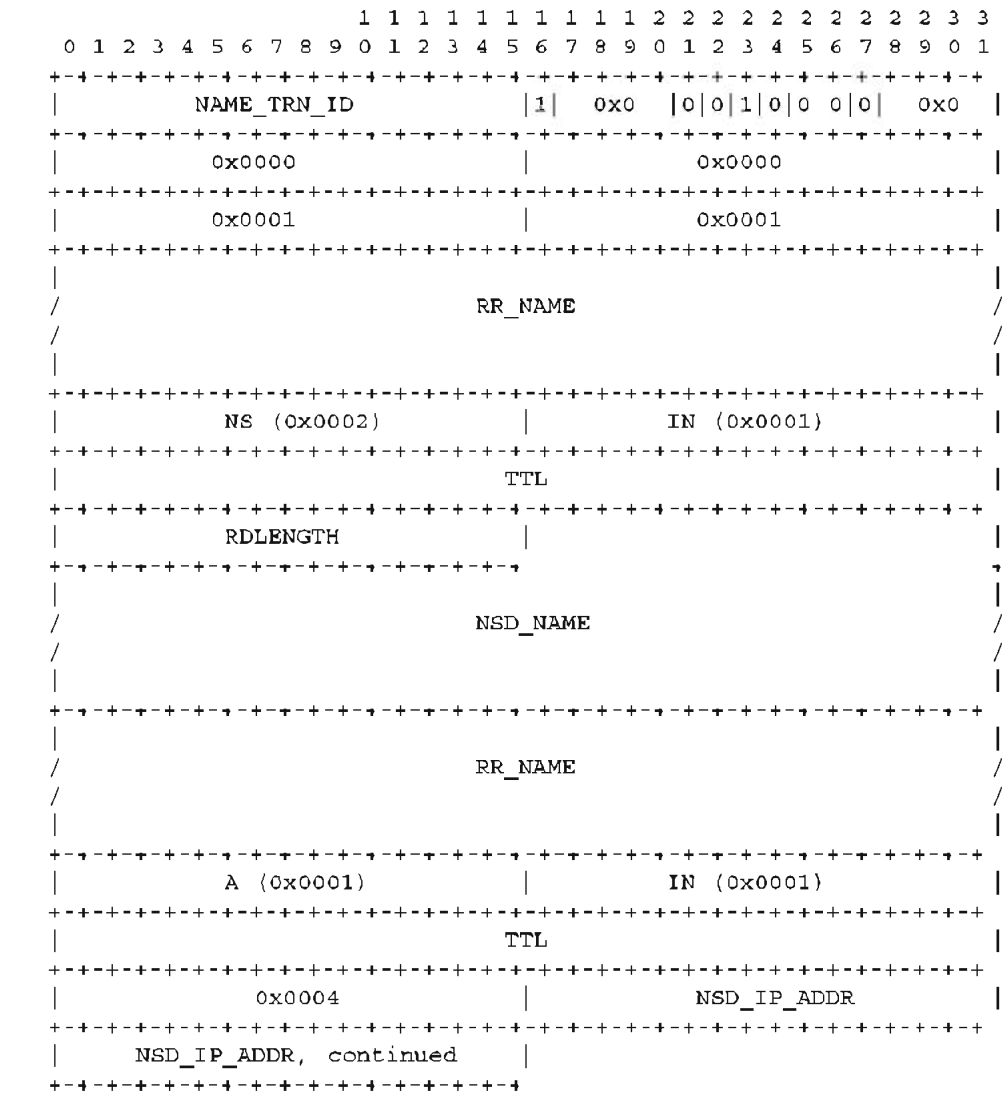
4.2.14. NEGATIVE NAME QUERY RESPONSE



RCODE field values:

Symbol	Value	Description
FMT_ERR	0x1	Format Error. Request was invalidly formatted.
SRV_ERR	0x2	Server failure. Problem with NBNS, cannot process name.
NAM_ERR	0x3	Name Error. The name requested does not exist.
IMP_ERR	0x4	Unsupported request error. Allowable only for challenging NBNS when gets an Update type registration request.
RFS_ERR	0x5	Refused error. For policy reasons server will not register this name from this host.

4.2.15. REDIRECT NAME QUERY RESPONSE



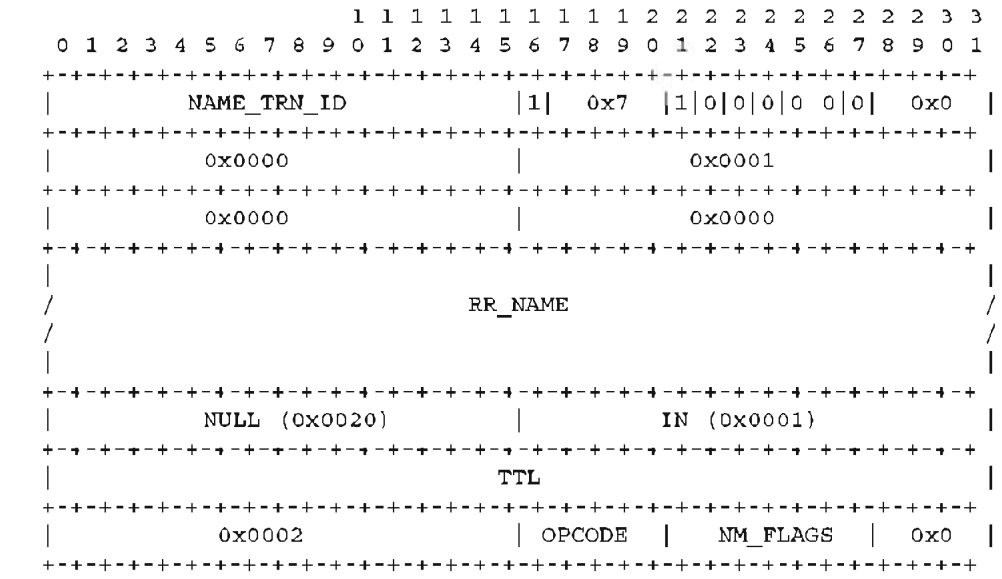
An end node responding to a NAME QUERY REQUEST always responds with the AA and RA bits set for both the NEGATIVE and POSITIVE NAME QUERY RESPONSE packets. An end node never sends a REDIRECT NAME QUERY RESPONSE packet.

When the requestor receives the REDIRECT NAME QUERY RESPONSE it must reiterate the NAME QUERY REQUEST to the NBNS specified by the NSD_IP_ADDR field of the A type RESOURCE RECORD in the ADDITIONAL section of the response packet. This is an optional packet for the NBNS.

The NSD_NAME and the RR_NAME in the ADDITIONAL section of the response packet are the same name. Space can be optimized if label string pointers are used in the RR_NAME which point to the labels in the NSD_NAME.

The RR_NAME in the AUTHORITY section is the name of the domain the NBNS called by NSD_NAME has authority over.

4.2.16. WAIT FOR ACKNOWLEDGEMENT (WACK) RESPONSE

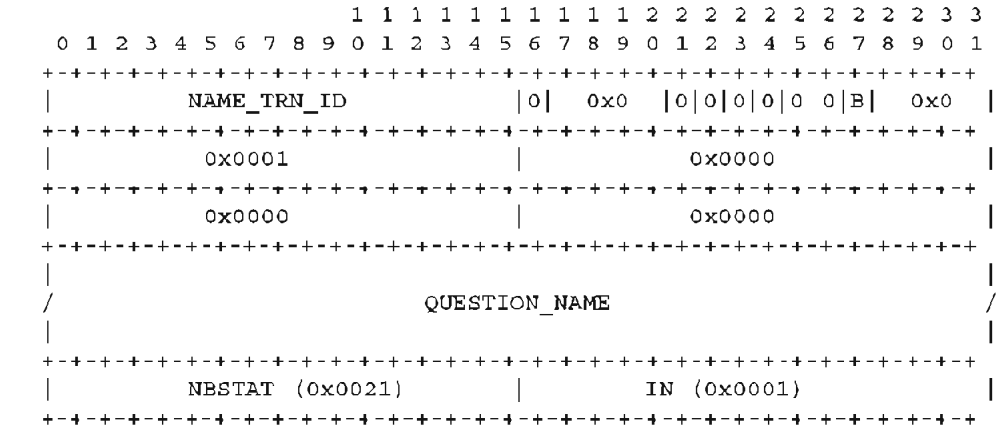


The NAME_TRN_ID of the WACK RESPONSE packet is the same NAME_TRN_ID of the request that the NBNS is telling the requestor to wait longer to complete. The RR_NAME is the name from the request, if any. If no name is available from the request then it is a null name, single byte of zero.

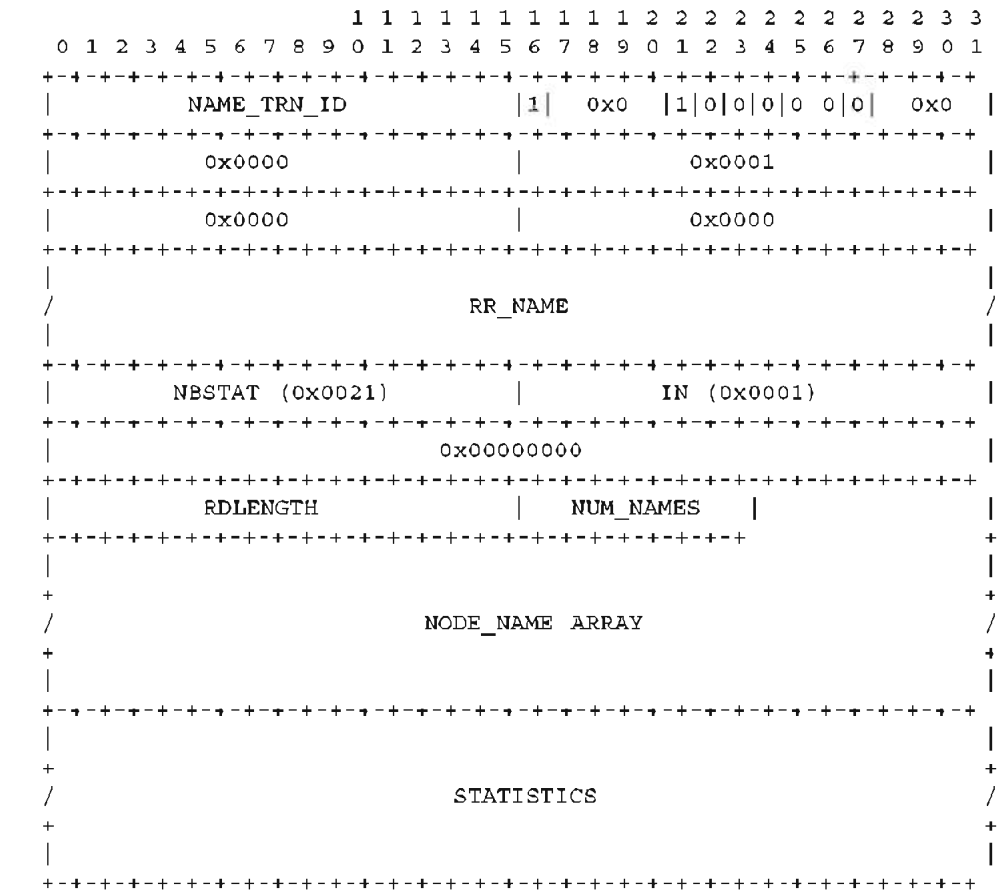
The TTL field of the ResourceRecord is the new time to wait, in seconds, for the request to complete. The RDATA field contains the OPCODE and NM_FLAGS of the request.

A TTL value of 0 means that the NBNS can not estimate the time it may take to complete a response.

4.2.17. NODE STATUS REQUEST

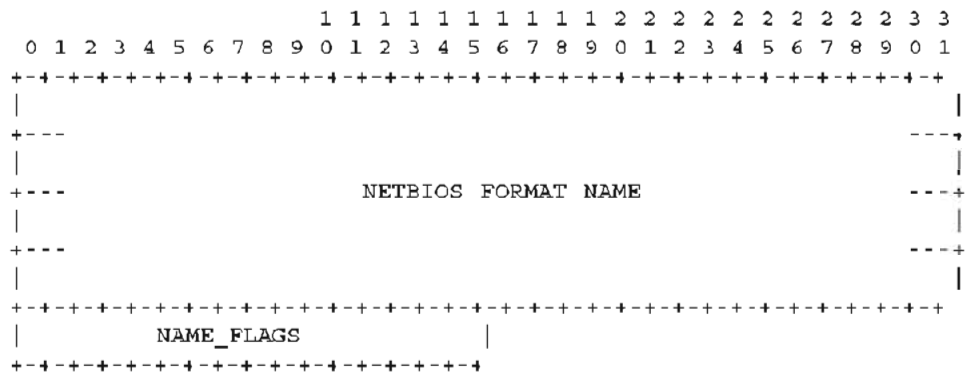


4.2.18. NODE STATUS RESPONSE

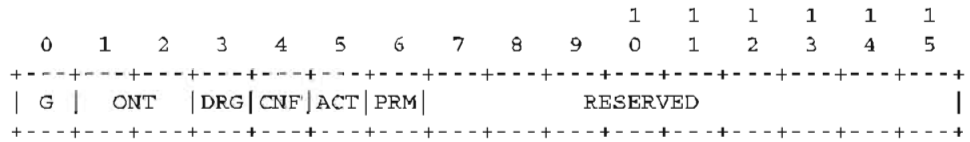


The NODE_NAME ARRAY is an array of zero or more NUM_NAMES entries of NODE_NAME records. Each NODE_NAME entry represents an active name in the same NetBIOS scope as the requesting name in the local name table of the responder. RR_NAME is the requesting name.

NODE_NAME Entry:



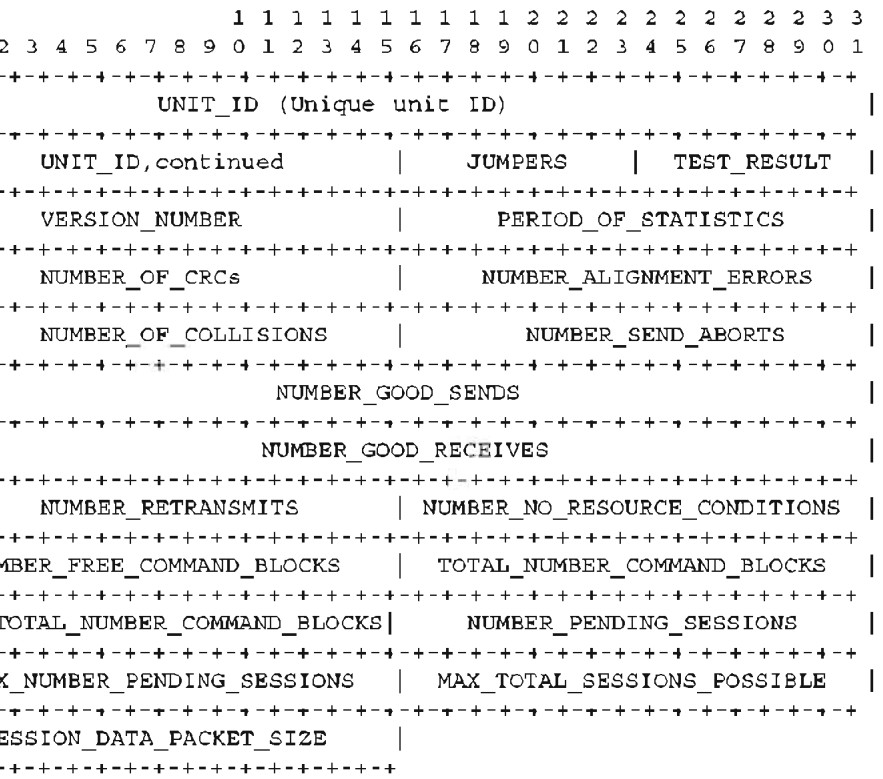
The NAME_FLAGS field:



The NAME_FLAGS field is defined as:

Symbol	Bit(s)	Description:
RESERVED	7-15	Reserved for future use. Must be zero (0).
PRM	6	Permanent Name Flag. If one (1) then entry is for the permanent node name. Flag is zero (0) for all other names.
ACT	5	Active Name Flag. All entries have this flag set to one (1).
CNF	4	Conflict Flag. If one (1) then name on this node is in conflict.
DRG	3	Deregister Flag. If one (1) then this name is in the process of being deleted.
ONT	1,2	Owner Node Type: 00 = B node 01 = P node 10 = M node 11 = Reserved for future use
G	0	Group Name Flag. If one (1) then the name is a GROUP NetBIOS name. If zero (0) then it is a UNIQUE NetBIOS name.

STATISTICS Field of the NODE STATUS RESPONSE:

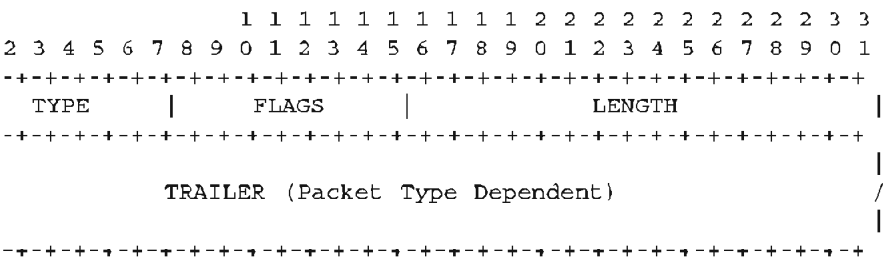


4.3. SESSION SERVICE PACKETS

4.3.1. GENERAL FORMAT OF SESSION PACKETS

All session service messages are sent over a TCP connection.

All session packets are of the following general structure:



The TYPE, FLAGS, and LENGTH fields are present in every session packet.

The LENGTH field is the number of bytes following the LENGTH field. In other words, LENGTH is the combined size of the TRAILER field(s). For example, the POSITIVE SESSION RESPONSE packet always has a LENGTH field value of zero (0000) while the RETARGET SESSION RESPONSE always has a LENGTH field value of six (0006).

One of the bits of the FLAGS field acts as an additional, high-order bit for the LENGTH field. Thus the cumulative size of the trailer field(s) may range from 0 to 128K bytes.

Session Packet Types (in hexidecimal):

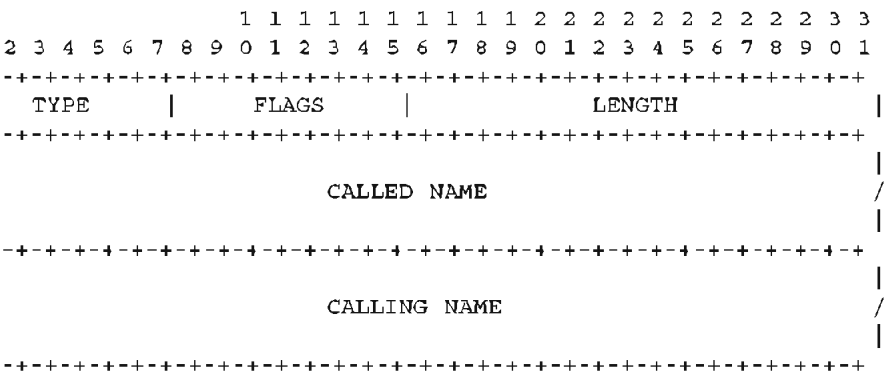
```
00 -  SESSION MESSAGE
81 -  SESSION REQUEST
82 -  POSITIVE SESSION RESPONSE
83 -  NEGATIVE SESSION RESPONSE
84 -  RETARGET SESSION RESPONSE
85 -  SESSION KEEP ALIVE
```

Bit definitions of the FLAGS field:

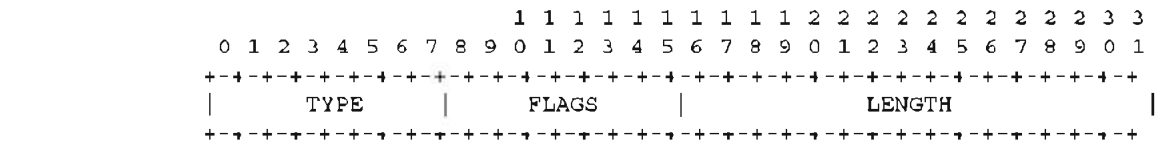
```
  0   1   2   3   4   5   6   7
+---+---+---+---+---+---+---+---+
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | E |
+---+---+---+---+---+---+---+---+
```

Symbol	Bit(s)	Description
E	7	Length extension, used as an additional, high-order bit on the LENGTH field.
RESERVED	0-6	Reserved, must be zero (0)

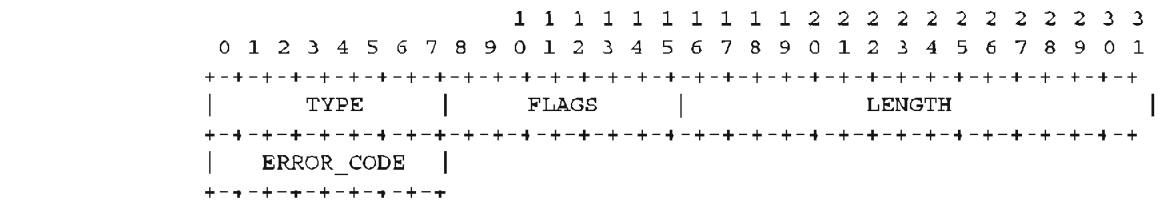
4.3.2. SESSION REQUEST PACKET



4.3.3. POSITIVE SESSION RESPONSE PACKET



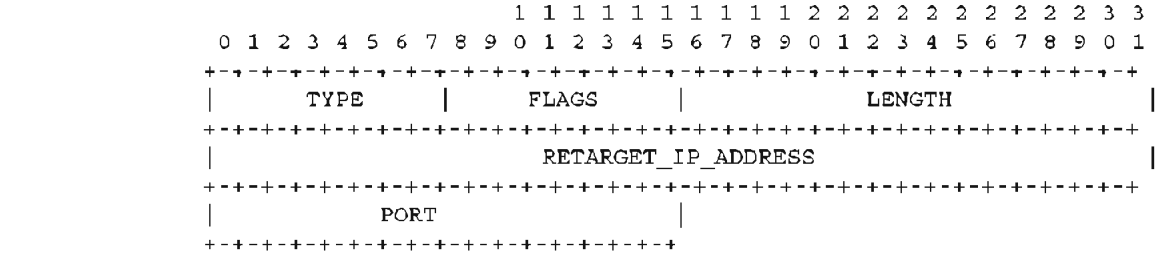
4.3.4. NEGATIVE SESSION RESPONSE PACKET



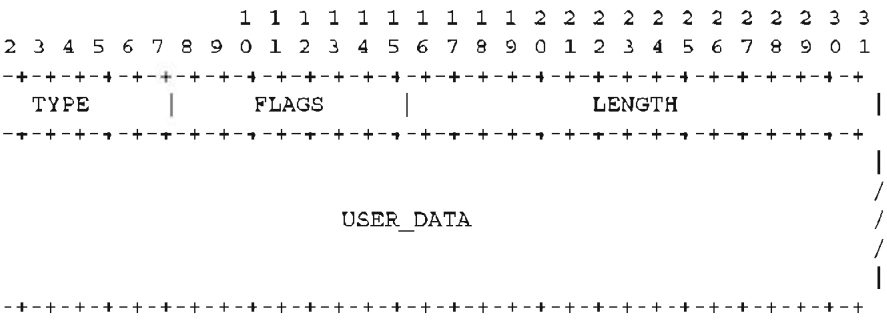
NEGATIVE SESSION RESPONSE packet error code values (in hexidecimal):

80 - Not listening on called name
81 - Not listening for calling name
82 - Called name not present
83 - Called name present, but insufficient resources
8F - Unspecified error

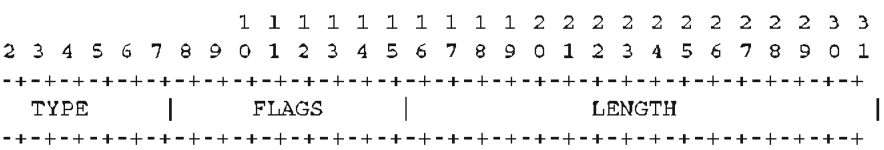
4.3.5. SESSION RETARGET RESPONSE PACKET



4.3.6. SESSION MESSAGE PACKET

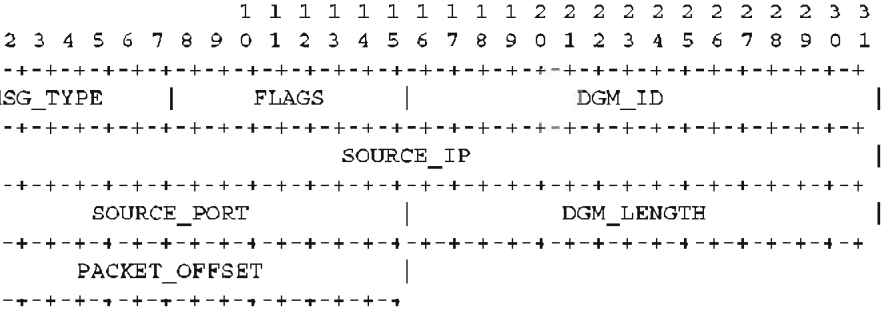


4.3.7. SESSION KEEP ALIVE PACKET



4.4. DATAGRAM SERVICE PACKETS

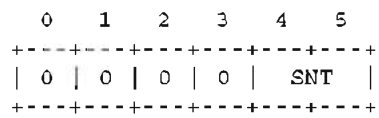
4.4.1. NetBIOS DATAGRAM HEADER



MSG_TYPE values (in hexidecimal):

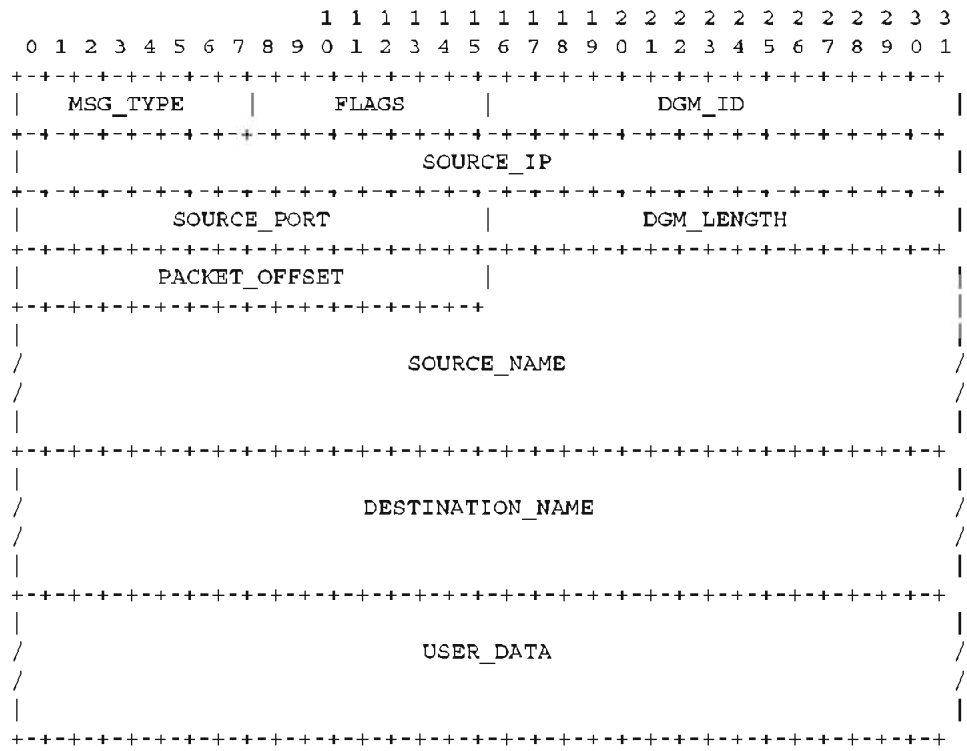
10 - DIRECT_UNIQUE DATAGRAM
11 - DIRECT_GROUP DATAGRAM
12 - BROADCAST DATAGRAM
13 - DATAGRAM ERROR
14 - DATAGRAM QUERY REQUEST
15 - DATAGRAM POSITIVE QUERY RESPONSE
16 - DATAGRAM NEGATIVE QUERY RESPONSE

Bit definitions of the FLAGS field:

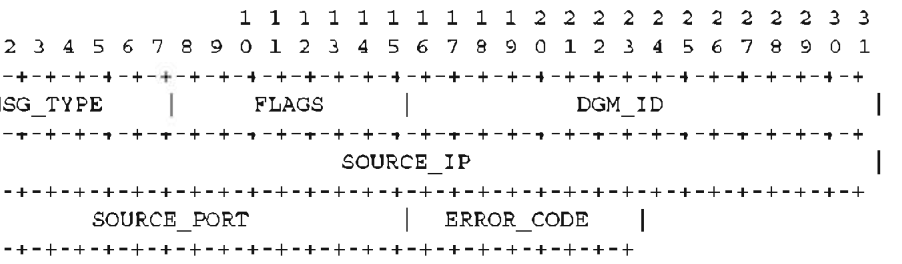


Symbol	Bit(s)	Description
M	7	MORE flag, If set then more NetBIOS datagram fragments follow.
F	6	FIRST packet flag, If set then this is first (and possibly only) fragment of NetBIOS datagram
SNT	4,5	Source End-Node type: 00 = B node 01 = P node 10 = M node 11 = NBDD
RESERVED	0-3	Reserved, must be zero (0)

4.4.2. DIRECT_UNIQUE, DIRECT_GROUP, & BROADCAST DATAGRAM



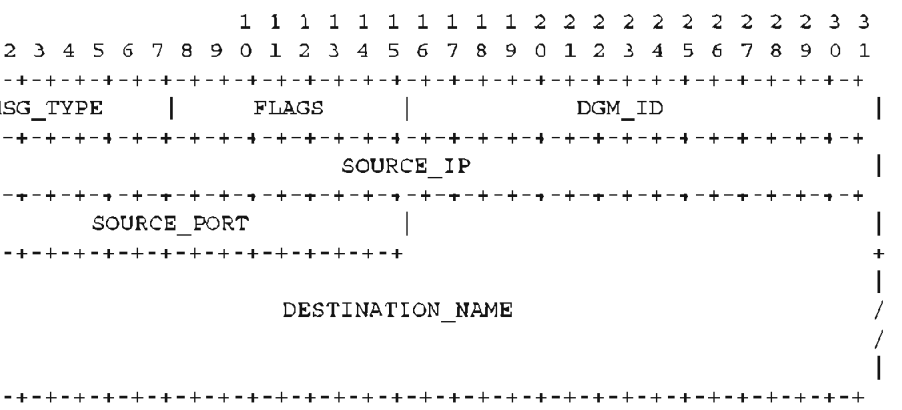
4.4.3. DATAGRAM ERROR PACKET



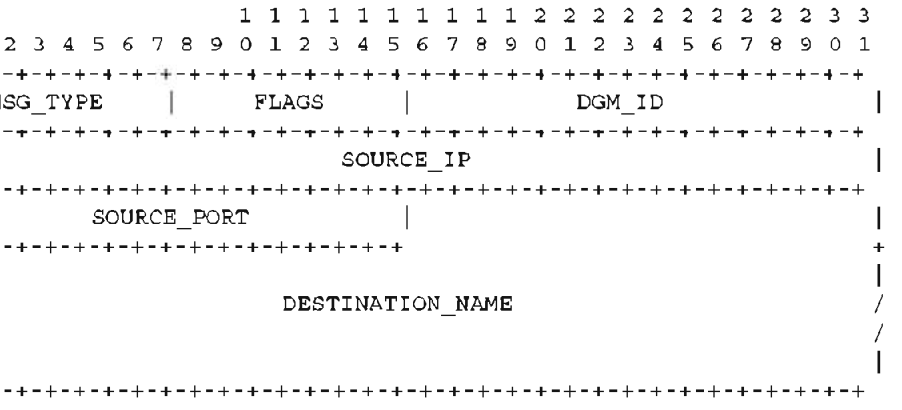
ERROR_CODE values (in hexidecimal):

82 - DESTINATION NAME NOT PRESENT
83 - INVALID SOURCE NAME FORMAT
84 - INVALID DESTINATION NAME FORMAT

4.4.4. DATAGRAM QUERY REQUEST



4.4.5. DATAGRAM POSITIVE AND NEGATIVE QUERY RESPONSE



5. PROTOCOL DESCRIPTIONS

5.1. NAME SERVICE PROTOCOLS

A REQUEST packet is always sent to the well known UDP port - NAME_SERVICE_UDP_PORT. The destination address is normally either the IP broadcast address or the address of the NBNS - the address of the NBNS server it set up at initialization time. In rare cases, a request packet will be sent to an end node, e.g. a NAME QUERY REQUEST sent to "challenge" a node.

A RESPONSE packet is always sent to the source UDP port and source IP address of the request packet.

A DEMAND packet must always be sent to the well known UDP port - NAME_SERVICE_UDP_PORT. There is no restriction on the target IP address.

Terms used in this section:

tid - Transaction ID. This is a value composed from the requestor's IP address and a unique 16 bit value generated by the originator of the transaction.

5.1.1. B-NODE ACTIVITY

5.1.1.1. B-NODE ADD NAME

```
PROCEDURE add_name(newname)

/*
 * Host initiated processing for a B node
 */
BEGIN

     REPEAT

          /* build name service packet */

          ONT = B_NODE; /* broadcast node */
          G = UNIQUE;   /* unique name */
          TTL = 0;

          broadcast NAME REGISTRATION REQUEST packet;

          /*
           * remote node(s) will send response packet
           * if applicable
           */
```

```
          pause(BCAST_REQ_RETRY_TIMEOUT);

     UNTIL response packet is received or
          retransmit count has been exceeded

     IF no response packet was received THEN
     BEGIN /* no response */
          /*
           * build packet
           */

          ONT = B_NODE; /* broadcast node */
          G = UNIQUE;   /* unique name */
          TTL = 0;

          /*
           * Let other nodes known you have the name
           */

          broadcast NAME UPDATE REQUEST packet;
          /* name can be added to local name table */
          return success;
     END /* no response */
     ELSE
     BEGIN /* got response */

          /*
           * Match return transaction id
           * against tid sent in request
           */

         IF NOT response tid = request tid THEN
         BEGIN
          ignore response packet;
         END
         ELSE
         CASE packet type OF

         NEGATIVE NAME REGISTRATION RESPONSE:

              return failure; /* name cannot be added */

         POSITIVE NAME REGISTRATION RESPONSE:
         END-NODE CHALLENGE NAME REGISTRATION RESPONSE:

              /*
               * B nodes should normally not get this
               * response.
               */

               ignore packet;
```

```
            END /* case */;
        END /* got response */
    END /* procedure */
```

5.1.1.2. B-NODE ADD_GROUP NAME

```
    PROCEDURE add_group_name(newname)

    /*
     * Host initiated processing for a B node
     */

    BEGIN
         /*
          * same as for a unique name with the
          * exception that the group bit (G) must
          * be set in the request packets.
          */

         ...
         G = GROUP;
         ...
         ...

         /*
          * broadcast request ...
          */


    END
```

5.1.1.3. B-NODE FIND_NAME

```
    PROCEDURE find_name(name)

    /*
     * Host initiated processing for a B node
     */

    BEGIN

         REPEAT
              /*
               * build packet
               */
              ONT = B;
              TTL = 0;
              G = DONT CARE;

              broadcast NAME QUERY REQUEST packet;
```

```
            /*
             * a node might send response packet
             */

            pause(BCAST_REQ_RETRY_TIMEOUT);
        UNTIL response packet received OR
            max transmit threshold exceeded

        IF no response packet received THEN
            return failure;
        ELSE
        IF NOT response tid = request tid THEN
            ignore packet;
        ELSE
        CASE packet type OF
        POSITIVE NAME QUERY RESPONSE:
            /*
             * Start a timer to detect conflict.
             *
             * Be prepared to detect conflict if
             * any more response packets are received.
             *
             */

            save response as authoritative response;
            start_timer(CONFLICT_TIMER);
            return success;

        NEGATIVE NAME QUERY RESPONSE:
        REDIRECT NAME QUERY RESPONSE:

            /*
             * B Node should normally not get either
             * response.
             */

             ignore response packet;

        END /* case */
    END /* procedure */
```

5.1.1.4. B NODE NAME RELEASE

```
    PROCEDURE delete_name (name)
    BEGIN

        REPEAT

            /*
             * build packet
             */
```

```
            ...

            /*
             * send request
             */

            broadcast NAME RELEASE REQUEST packet;

            /*
             * no response packet expected
             */

            pause(BCAST_REQ_RETRY_TIMEOUT);

      UNTIL retransmit count has been exceeded
END /* procedure */
```

5.1.1.5. B-NODE INCOMING PACKET PROCESSING

Following processing is done when broadcast or unicast packets are received at the NAME_SERVICE_UDP_PORT.

```
PROCEDURE process_incoming_packet(packet)

/*
 * Processing initiated by incoming packets for a B node
 */

BEGIN
     /*
      * Note: response packets are always sent
      * to:
      * source IP address of request packet
      * source UDP port of request packet
      */

     CASE packet type OF

     NAME REGISTRATION REQUEST (UNIQUE):
          IF name exists in local name table THEN
               send NEGATIVE NAME REGISTRATION RESPONSE ;
     NAME REGISTRATION REQUEST (GROUP):
          IF name exists in local name table THEN
          BEGIN
               IF local entry is a unique name THEN
                    send NEGATIVE NAME REGISTRATION RESPONSE ;
          END
     NAME QUERY REQUEST:
          IF name exists in local name table THEN
          BEGIN
               build response packet;
```

```
            send POSITIVE NAME QUERY RESPONSE;
      POSITIVE NAME QUERY RESPONSE:
           IF name conflict timer is not active THEN
           BEGIN
                /*
                 * timer has expired already...  ignore this
                 * packet
                 */

                return;
           END
           ELSE /* timer is active */
           IF a response for this name has previously been
                received THEN
           BEGIN /* existing entry */

                /*
                 * we sent out a request packet, and
                 * have already received (at least)
                 * one response
                 *
                 * Check if conflict exists.
                 * If so, send out a conflict packet.
                 *
                 * Note: detecting conflict does NOT
                 * affect any existing sessions.
                 *
                 */

                /*
                 * Check for name conflict.
                 * See "Name Conflict" in Concepts and Methods
                 */
                check saved authoritative response against
                     information in this response packet;
                IF conflict detected THEN
                BEGIN
                     unicast NAME CONFLICT DEMAND packet;
                     IF entry exists in cache THEN
                     BEGIN
                          remove entry from cache;
                     END
                END
           END /* existing entry */
           ELSE
           BEGIN
                /*
                 * Note: If this was the first response
                 * to a name query, it would have been
                 * handled in the
                 * find_name() procedure.
```

```
                */

                ignore packet;
          END
     NAME CONFLICT DEMAND:
          IF name exists in local name table THEN
          BEGIN
                mark name as conflict detected;

                /*
                 * a name in the state "conflict detected"
                 * does not "logically" exist on that node.
                 * No further session will be accepted on
                 * that name.
                 * No datagrams can be sent against that name.
                 * Such an entry will not be used for
                 * purposes of processing incoming request
                 * packets.
                 * The only valid user NetBIOS operation
                 * against such a name is DELETE NAME.
                 */
          END
     NAME RELEASE REQUEST:
          IF caching is being done THEN
          BEGIN
                remove entry from cache;
          END
     NAME UPDATE REQUEST:
          IF caching is being done THEN
          BEGIN
                IF entry exists in cache already,
                     update cache;
                ELSE IF name is "interesting" THEN
                BEGIN
                     add entry to cache;
                END
          END

     NODE STATUS REQUEST:
          IF name exists in local name table THEN
          BEGIN
                /*
                 * send only those names that are
                 * in the same scope as the scope
                 * field in the request packet
                 */

                send NODE STATUS RESPONSE;
          END
END
```

5.1.2. P-NODE ACTIVITY

All packets sent or received by P nodes are unicast UDP packets. A P node sends name service requests to the NBNS node that is specified in the P-node configuration.

5.1.2.1. P-NODE ADD_NAME

```
PROCEDURE add_name(newname)

/*
 * Host initiated processing for a P node
 */

BEGIN

     REPEAT
          /*
           * build packet
           */

          ONT = P;
          G = UNIQUE;
          ...

          /*
           * send request
           */

          unicast NAME REGISTRATION REQUEST packet;

          /*
           * NBNS will send response packet
           */

          IF receive a WACK RESPONSE THEN
               pause(time from TTL field of response);
          ELSE
               pause(UCAST_REQ_RETRY_TIMEOUT);
     UNTIL response packet is received OR
          retransmit count has been exceeded

     IF no response packet was received THEN
     BEGIN /* no response */
          /*
           * NBNS is down.  Cannot claim name.
           */

          return failure; /* name cannot be claimed */
     END /* no response */
     ELSE
```

```
BEGIN /* response */
    IF NOT response tid = request tid THEN
    BEGIN
     /*  Packet may belong to another transaction  */
     ignore response packet;
    END
    ELSE
    CASE packet type OF

    POSITIVE NAME REGISTRATION RESPONSE:

         /*
          * name can be added
          */

         adjust refresh timeout value, TTL, for this name;
         return success;       /* name can be added */

    NEGATIVE NAME REGISTRATION RESPONSE:
         return failure; /* name cannot be added */

    END-NODE CHALLENGE REGISTRATION REQUEST:
    BEGIN /* end node challenge */

         /*
          * The response packet has in it the
          * address of the presumed owner of the
          * name.  Challenge that owner.
          * If owner either does not
          * respond or indicates that he no longer
          * owns the name, claim the name.
          * Otherwise, the name cannot be claimed.
          *
          */

         REPEAT
          /*
           * build packet
           */
          ...

          unicast NAME QUERY REQUEST packet to the
               address contained in the END NODE
               CHALLENGE RESPONSE packet;

          /*
           * remote node may send response packet
           */

          pause(UCAST_REQ_RETRY_TIMEOUT);
```

```
UNTIL response packet is received or
    retransmit count has been exceeded
IF no response packet is received OR
      NEGATIVE NAME QUERY RESPONSE packet
      received THEN
BEGIN /* update */

 /*
  * name can be claimed
  */

 REPEAT

     /*
      * build packet
      */
      ...

     unicast NAME UPDATE REQUEST to NBNS;

     /*
      * NBNS node will send response packet
      */

     IF receive a WACK RESPONSE THEN
           pause(time from TTL field of response);
     ELSE
           pause(UCAST_REQ_RETRY_TIMEOUT);
 UNTIL response packet is received or
     retransmit count has been exceeded
 IF no response packet received THEN
 BEGIN /* no response */

      /*
       * name could not be claimed
       */

      return failure;
 END /* no response */
 ELSE
 CASE packet type OF
      POSITIVE NAME REGISTRATION RESPONSE:
           /*
            * add name
            */
           return success;
      NEGATIVE NAME REGISTRATION RESPONSE:

           /*
            * you lose  ...
            */
```

```
                        return failure;
                    END /* case */
                END /* update */
                ELSE

                /*
                 * received a positive response to the "challenge"
                 * Remote node still has name
                 */

                 return failure;
            END /* end node challenge */
        END /* response */
    END /* procedure */
```

5.1.2.2. P-NODE ADD GROUP NAME

```
    PROCEDURE add_group_name(newname)

    /*
     * Host initiated processing for a P node
     */

    BEGIN
        /*
         * same as for a unique name, except that the
         * request packet must indicate that a
         * group name claim is being made.
         */

        ...
        G = GROUP;
        ...

        /*
         * send packet
         */
         ...


    END
```

5.1.2.3. P-NODE FIND NAME

```
    PROCEDURE find_name(name)

    /*
     * Host initiated processing for a P node
     */

    BEGIN
```

```
        REPEAT
             /*
              * build packet
              */

             ONT = P;
             G = DONT CARE;

             unicast NAME QUERY REQUEST packet;

             /*
              * a NBNS node might send response packet
              */

             IF receive a WACK RESPONSE THEN
                  pause(time from TTL field of response);
             ELSE
                  pause(UCAST_REQ_RETRY_TIMEOUT);
        UNTIL response packet received OR
             max transmit threshold exceeded

        IF no response packet received THEN
             return failure;
        ELSE
        IF NOT response tid = request tid THEN
             ignore packet;
        ELSE
        CASE packet type OF
        POSITIVE NAME QUERY RESPONSE:
             return success;

        REDIRECT NAME QUERY RESPONSE:

             /*
              * NBNS node wants this end node
              * to use some other NBNS node
              * to resolve the query.
              */

              repeat query with NBNS address
                  in the response packet;
        NEGATIVE NAME QUERY RESPONSE:
             return failure;

        END /* case */
   END /* procedure */
```

5.1.2.4. P-NODE DELETE_NAME

```
   PROCEDURE delete_name (name)
```

```
   /*
    * Host initiated processing for a P node
    */

   BEGIN

        REPEAT

             /*
              * build packet
              */
             ...

             /*
              * send request
              */

             unicast NAME RELEASE REQUEST packet;
             IF receive a WACK RESPONSE THEN
                  pause(time from TTL field of response);
             ELSE
                  pause(UCAST_REQ_RETRY_TIMEOUT);
        UNTIL retransmit count has been exceeded
             or response been received

        IF response has been received THEN
        CASE packet type OF
        POSITIVE NAME RELEASE RESPONSE:
             return success;
        NEGATIVE NAME RELEASE RESPONSE:

             /*
              * NBNS does want node to delete this
              * name !!!
              */

             return failure;
        END /* case */
   END /* procedure */
```

5.1.2.5. P-NODE INCOMING PACKET PROCESSING

```
   Processing initiated by reception of packets at a P node

   PROCEDURE process_incoming_packet(packet)

   /*
    * Processing initiated by incoming packets at a P node
    */

   BEGIN
```

```
/*
 * always ignore UDP broadcast packets
 */

IF packet was sent as a broadcast THEN
BEGIN
     ignore packet;
     return;
END
CASE packet type of

NAME CONFLICT DEMAND:
     IF name exists in local name table THEN
          mark name as in conflict;
     return;

NAME QUERY REQUEST:
     IF name exists in local name table THEN
     BEGIN /* name exists */

          /*
           * build packet
           */
          ...

          /*
           * send response to the IP address and port
           * number from which the request was received.
           */

          send POSITIVE NAME QUERY RESPONSE ;
          return;
     END /* exists */
     ELSE
     BEGIN /* does not exist */

          /*
           * send response to the requestor
           */

          send NEGATIVE NAME QUERY RESPONSE ;
          return;
     END /* does not exist */
NODE STATUS REQUEST:
     /*
      * Name of "*" may be used for force node to
      * divulge status for administrative purposes
      */
     IF name in local name table OR name = "*" THEN
     BEGIN
          /*
```

```
                    * Build response packet and
                    * send to requestor node
                    * Send only those names that are
                    * in the same scope as the scope
                    * in the request packet.
                    */

                    send NODE STATUS RESPONSE;
               END

          NAME RELEASE REQUEST:
               /*
                * This will be received if the NBNS wants to flush the
                * name from the local name table, or from the local
                * cache.
                */

               IF name exists in the local name table THEN
               BEGIN
                    delete name from local name table;
                    inform user that name has been deleted;
               END
               ELSE
                    IF name has been cached locally THEN
                    BEGIN
                         remove entry from cache:
                    END

          END /* case */
     END /* procedure */
```

5.1.2.6. P-NODE TIMER INITIATED PROCESSING

Processing initiated by timer expiration.

```
PROCEDURE timer_expired()
/*
 * Processing initiated by the expiration of a timer on a P node
 */
BEGIN
     /*
      * Send a NAME REFRESH REQUEST for each name which the
      * TTL which has expired.
      */
     REPEAT
          build NAME REFRESH REQUEST packet;
          REPEAT
               send packet to NBNS;

               IF receive a WACK RESPONSE THEN
                    pause(time from TTL field of response);
```

```
                ELSE
                     pause(UCAST_REQ_RETRY_TIMEOUT);
           UNTIL response packet is received or
                retransmit count has been exceeded

           CASE packet type OF
             POSITIVE NAME REGISTRATION RESPONSE:
                /* successfully refreshed */
                reset TTL timer for this name;

             NEGATIVE NAME REGISTRATION RESPONSE:
                /*
                 * refused, can't keep name
                 * assume in conflict
                 */
                mark name as in conflict;
           END /* case */

      UNTIL request sent for all names for which TTL
           has expired
END /* procedure */
```

5.1.3. M-NODE ACTIVITY

M nodes behavior is similar to that of P nodes with the addition of some B node-like broadcast actions. M node name service proceeds in two steps:

1. Use broadcast UDP based name service. Depending on the operation, goto step 2.

2. Use directed UDP name service.

The following code for M nodes is exactly the same as for a P node, with the exception that broadcast operations are done before P type operation is attempted.

5.1.3.1. M-NODE ADD NAME

```
PROCEDURE add_name(newname)

/*
 * Host initiated processing for a M node
 */

BEGIN

     /*
      * check if name exists on the
      * broadcast area
      */
```

```
REPEAT
    /* build packet */

    ....
    broadcast NAME REGISTRATION REQUEST packet;
    pause(BCAST_REQ_RETRY_TIMEOUT);

UNTIL response packet is received or
     retransmit count has been  exceeded

IF valid response received THEN
BEGIN
     /* cannot claim name */

     return failure;
END

/*
 * No objections received within the
 * broadcast area.
 * Send request to name server.
 */

REPEAT
     /*
      * build packet
      */

     ONT = M;
     ...

     unicast NAME REGISTRATION REQUEST packet;

     /*
      * remote NBNS will send response packet
      */

     IF receive a WACK RESPONSE THEN
          pause(time from TTL field of response);
     ELSE
          pause(UCAST_REQ_RETRY_TIMEOUT);

UNTIL response packet is received or
     retransmit count has been exceeded

IF no response packet was received THEN
BEGIN /* no response */
     /*
      * NBNS is down.  Cannot claim name.
      */
```

```
     return failure; /* name cannot be claimed */
END /* no response */
ELSE
BEGIN /* response */
    IF NOT response tid = request tid THEN
    BEGIN
     ignore response packet;
    END
    ELSE
    CASE packet type OF
    POSITIVE NAME REGISTRATION RESPONSE:

         /*
          * name can be added
          */

         adjust refresh timeout value, TTL;
         return success;       /* name can be added */

    NEGATIVE NAME REGISTRATION RESPONSE:
         return failure; /* name cannot be added */

    END-NODE CHALLENGE REGISTRATION REQUEST:
    BEGIN /* end node challenge */

         /*
          * The response packet has in it the
          * address of the presumed owner of the
          * name.  Challenge that owner.
          * If owner either does not
          * respond or indicates that he no longer
          * owns the name, claim the name.
          * Otherwise, the name cannot be claimed.
          *
          */

         REPEAT
          /*
           * build packet
           */
          ...

          /*
           * send packet to address contained in the
           * response packet
           */

          unicast NAME QUERY REQUEST packet;

          /*
           * remote node may send response packet
```

```
 */

pause(UCAST_REQ_RETRY_TIMEOUT);

UNTIL response packet is received or
    retransmit count has been exceeded
IF no response packet is received THEN
BEGIN /* no response */

 /*
  * name can be claimed
  */
 REPEAT

     /*
      * build packet
      */
      ...

     unicast NAME UPDATE REQUEST to NBNS;

     /*
      * NBNS node will send response packet
      */

     IF receive a WACK RESPONSE THEN
          pause(time from TTL field of response);
 ELSE
      pause(UCAST_REQ_RETRY_TIMEOUT);

 UNTIL response packet is received or
     retransmit count has been exceeded
 IF no response packet received THEN
 BEGIN /* no response */

      /*
       * name could not be claimed
       */

      return failure;
 END /* no response */
 ELSE
 CASE packet type OF
 POSITIVE NAME REGISTRATION RESPONSE:
      /*
       * add name
       */

      return success;
 NEGATIVE NAME REGISTRATION RESPONSE:
```

```
                    /*
                     * you lose  ...
                     */

                    return failure;
                 END /* case */
                END /* no response */
                ELSE
                IF NOT response tid = request tid THEN
                BEGIN
                 ignore response packet;
                END

                /*
                 * received a response to the "challenge"
                 * packet
                 */

                CASE packet type OF
                POSITIVE NAME QUERY:

                 /*
                  * remote node still has name.
                  */

                 return failure;
                NEGATIVE NAME QUERY:

                 /*
                  * remote node no longer has name
                  */

                 return success;
                END /* case */
            END /* end node challenge */
            END /* case */
        END /* response */
   END /* procedure */
```

5.1.3.2. M-NODE ADD GROUP NAME

```
   PROCEDURE add_group_name(newname)

   /*
    * Host initiated processing for a P node
    */

   BEGIN
        /*
         * same as for a unique name, except that the
         * request packet must indicate that a
```

```
         * group name claim is being made.
         */

        ...
        G = GROUP;
        ...

        /*
         * send packet
         */
         ...


   END
```

5.1.3.3. M-NODE FIND NAME

```
   PROCEDURE find_name(name)

   /*
    * Host initiated processing for a M node
    */

   BEGIN
        /*
         * check if any node on the broadcast
         * area has the name
         */

        REPEAT
             /* build packet */
             ...

             broadcast NAME QUERY REQUEST packet;
             pause(BCAST_REQ_RETRY_TIMEOUT);
        UNTIL response packet received OR
             max transmit threshold exceeded

        IF valid response received THEN
        BEGIN
             save response as authoritative response;
             start_timer(CONFLICT_TIMER);
             return success;
        END

        /*
         * no valid response on the b'cast segment.
         * Try the name server.
         */

        REPEAT
```

```
                /*
                 * build packet
                 */

                ONT = M;
                G = DONT CARE;

                unicast NAME QUERY REQUEST packet to NBNS;

                /*
                 * a NBNS node might send response packet
                 */

                IF receive a WACK RESPONSE THEN
                     pause(time from TTL field of response);
                ELSE
                     pause(UCAST_REQ_RETRY_TIMEOUT);
           UNTIL response packet received OR
                max transmit threshold exceeded

           IF no response packet received THEN
                return failure;
           ELSE
           IF NOT response tid = request tid THEN
                ignore packet;
           ELSE
           CASE packet type OF
           POSITIVE NAME QUERY RESPONSE:
                return success;

           REDIRECT NAME QUERY RESPONSE:

                /*
                 * NBNS node wants this end node
                 * to use some other NBNS node
                 * to resolve the query.
                 */

                 repeat query with NBNS address
                     in the response packet;
           NEGATIVE NAME QUERY RESPONSE:
                return failure;

           END /* case */
      END /* procedure */
```

5.1.3.4. M-NODE DELETE NAME

```
   PROCEDURE delete_name (name)

   /*
```

```
    * Host initiated processing for a P node
    */

   BEGIN
        /*
         * First, delete name on NBNS
         */

        REPEAT

             /*
              * build packet
              */
             ...

             /*
              * send request
              */

             unicast NAME RELEASE REQUEST packet to NBNS;

             IF receive a WACK RESPONSE THEN
                  pause(time from TTL field of response);
             ELSE
                  pause(UCAST_REQ_RETRY_TIMEOUT);
        UNTIL retransmit count has been exceeded
             or response been received

        IF response has been received THEN
        CASE packet type OF
        POSITIVE NAME RELEASE RESPONSE:
             /*
              * Deletion of name on b'cast segment is deferred
              * until after NBNS has deleted the name
              */

             REPEAT
                  /* build packet */

                  ...
                  broadcast NAME RELEASE REQUEST;
                  pause(BCAST_REQ_RETRY_TIMEOUT);
             UNTIL rexmt threshold exceeded

             return success;
        NEGATIVE NAME RELEASE RESPONSE:

             /*
              * NBNS does want node to delete this
              * name
              */
```

```
            return failure;
        END /* case */
    END /* procedure */
```

5.1.3.5. M-NODE INCOMING PACKET PROCESSING

Processing initiated by reception of packets at a M node

```
    PROCEDURE process_incoming_packet(packet)

    /*
     * Processing initiated by incoming packets at a M node
     */

    BEGIN
         CASE packet type of

         NAME CONFLICT DEMAND:
              IF name exists in local name table THEN
                   mark name as in conflict;
              return;

         NAME QUERY REQUEST:
              IF name exists in local name table THEN
              BEGIN /* name exists */

                   /*
                    * build packet
                    */
                   ...

                   /*
                    * send response to the IP address and port
                    * number from which the request was received.
                    */

                   send POSITIVE NAME QUERY RESPONSE ;
                   return;
              END /* exists */
              ELSE
              BEGIN /* does not exist */

                   /*
                    * send response to the requestor
                    */

                   IF request NOT broadcast THEN
                        /*
                         * Don't send negative responses to
                         * queries sent by B nodes
                         */
```

```
                    send NEGATIVE NAME QUERY RESPONSE ;
                return;
            END /* does not exist */
        NODE STATUS REQUEST:
            BEGIN
            /*
             * Name of "*" may be used for force node to
             * divulge status for administrative purposes
             */
            IF name in local name table OR name = "*" THEN
                /*
                 * Build response packet and
                 * send to requestor node
                 * Send only those names that are
                 * in the same scope as the scope
                 * in the request packet.
                 */

                send NODE STATUS RESPONSE;
            END

        NAME RELEASE REQUEST:
            /*
             * This will be received if the NBNS wants to flush the
             * name from the local name table, or from the local
             * cache.
             */

            IF name exists in the local name table THEN
            BEGIN
                delete name from local name table;
                inform user that name has been deleted;
            END
            ELSE
                IF name has been cached locally THEN
                BEGIN
                    remove entry from cache:
                END

        NAME REGISTRATION REQUEST (UNIQUE):
            IF name exists in local name table THEN
                send NEGATIVE NAME REGISTRATION RESPONSE ;
        NAME REGISTRATION REQUEST (GROUP):
            IF name exists in local name table THEN
            BEGIN
                IF local entry is a unique name THEN
                    send NEGATIVE NAME REGISTRATION RESPONSE ;
            END
        END /* case */
    END /* procedure */
```

5.1.3.6. M-NODE TIMER INITIATED PROCESSING

Processing initiated by timer expiration:

```
PROCEDURE timer_expired()
/*
 * Processing initiated by the expiration of a timer on a M node
 */
BEGIN
     /*
      * Send a NAME REFRESH REQUEST for each name which the
      * TTL which has expired.
      */
     REPEAT
          build NAME REFRESH REQUEST packet;
          REPEAT
               send packet to NBNS;

               IF receive a WACK RESPONSE THEN
                    pause(time from TTL field of response);
               ELSE
                    pause(UCAST_REQ_RETRY_TIMEOUT);
          UNTIL response packet is received or
               retransmit count has been exceeded

          CASE packet type OF
            POSITIVE NAME REGISTRATION RESPONSE:
               /* successfully refreshed */
               reset TTL timer for this name;

            NEGATIVE NAME REGISTRATION RESPONSE:
               /*
                * refused, can't keep name
                * assume in conflict
                */
               mark name as in conflict;
          END /* case */

     UNTIL request sent for all names for which TTL
          has expired
END /* procedure */
```

5.1.4. NBNS ACTIVITY

A NBNS node will receive directed packets from P and M nodes. Reply packets are always sent as directed packets to the source IP address and UDP port number. Received broadcast packets must be ignored.

5.1.4.1. NBNS INCOMING PACKET PROCESSING

```
PROCEDURE process_incoming_packet(packet)

/*
 * Incoming packet processing on a NS node
 */

BEGIN
     IF packet was sent as a broadcast THEN
     BEGIN
          discard packet;
          return;
     END
     CASE packet type of

     NAME REGISTRATION REQUEST (UNIQUE):
          IF unique name exists in data base THEN
          BEGIN /* unique name exists */
               /*
                * NBNS node may be a "passive"
                * server in that it expects the
                * end node to do the challenge
                * server.  Such a NBNS node is
                * called a "non-secure" server.
                * A "secure" server will do the
                * challenging before it sends
                * back a response packet.
                */

               IF non-secure THEN
               BEGIN
                    /*
                     * build response packet
                     */
                    ...


                    /*
                     * let end node do the challenge
                     */

                    send END-NODE CHALLENGE NAME REGISTRATION
                         RESPONSE;
                    return;
               END
               ELSE
               /*
                * secure server - do the name
                * challenge operation
                */
```

```
REPEAT
    send NAME QUERY REQUEST;
    pause(UCAST_REQ_RETRY_TIMEOUT);
UNTIL response has been received or
     retransmit count has been exceeded
IF no response was received THEN
BEGIN

     /* node down */

     update data base - remove entry;
     update data base - add new entry;
     send POSITIVE NAME REGISTRATION RESPONSE;
     return;
END
ELSE
BEGIN /* challenged node replied */
    /*
     * challenged node replied with
     * a response packet
     */

    CASE packet type

    POSITIVE NAME QUERY RESPONSE:

     /*
      * name still owned by the
      * challenged node
      *
      * build packet and send response
      */
      ...


     /*
      * Note: The NBNS will need to
      * keep track (based on transaction id) of
      * the IP address and port number
      * of the original requestor.
      */

     send NEGATIVE NAME REGISTRATION RESPONSE;
     return;
    NEGATIVE NAME QUERY RESPONSE:

     update data base - remove entry;
     update data base - add new  entry;

     /*
      * build response packet and send
```

```
                    * response
                    */
                   send POSITIVE NAME REGISTRATION RESPONSE;
                   return;
                  END /* case */
             END /* challenged node replied */
        END /* unique name exists in data base */
        ELSE
        IF group name exists in data base THEN
        BEGIN /* group names exists */

             /*
              * Members of a group name are NOT
              * challenged.
              * Make the assumption that
              * at least some of the group members
              * are still alive.
              * Refresh mechanism will
              * allow the NBNS to detect when all
              * members of a group no longer use that
              * name
              */

              send NEGATIVE NAME REGISTRATION RESPONSE;
        END /* group name exists */
        ELSE
        BEGIN /* name does not exist */

             /*
              * Name does not exist in data base
              *
              * This code applies to both non-secure
              * and secure server.
              */

             update data base - add new entry;
             send POSITIVE NAME REGISTRATION RESPONSE;
             return;
        END

   NAME QUERY REQUEST:
        IF name exists in data base THEN
        BEGIN
             /*
              * build response packet and send to
              * requestor
              */
              ...

             send POSITIVE NAME QUERY RESPONSE;
             return;
```

```
            ELSE
            BEGIN
                 /*
                  * build response packet and send to
                  * requestor
                  */
                  ...

                 send NEGATIVE NAME QUERY RESPONSE;
                 return;
            END

       NAME REGISTRATION REQUEST (GROUP):
            IF name exists in data base THEN
            BEGIN
                 IF local entry is a unique name THEN
                 BEGIN /* local is unique */

                     IF non-secure THEN
                     BEGIN
                      send  END-NODE CHALLENGE NAME
                           REGISTRATION RESPONSE;
                      return;
                     END

                     REPEAT
                      send NAME QUERY REQUEST;
                      pause(UCAST_REQ_RETRY_TIMEOUT);
                     UNTIL response received or
                          retransmit count exceeded
                     IF no response received or
                          NEGATIVE NAME QUERY RESPONSE
                           received THEN
                     BEGIN
                      update data base - remove entry;
                      update data base - add new entry;
                      send POSITIVE NAME REGISTRATION RESPONSE;
                      return;
                     END
                     ELSE
                     BEGIN
                      /*
                       * name still being held
                       * by challenged node
                       */

                       send NEGATIVE NAME REGISTRATION RESPONSE;
                     END
                 END /* local is unique */
                 ELSE
                 BEGIN /* local is group  */
```

```
                    /*
                     * existing entry is a group name
                     */

                    update data base - remove entry;
                    update data base - add new entry;
                    send POSITIVE NAME REGISTRATION RESPONSE;
                    return;
               END /* local is group */
          END /* names exists */
          ELSE
          BEGIN /* does not exist */

               /* name does not exist in data base */

               update data base - add new entry;
               send POSITIVE NAME REGISTRATION RESPONSE;
               return;
          END /* does not exist */

     NAME RELEASE REQUEST:

          /*
           * secure server may choose to disallow
           * a node from deleting a name
           */

          update data base - remove entry;
          send POSITIVE NAME RELEASE RESPONSE;
          return;

     NAME UPDATE REQUEST:

          /*
           * End-node completed a successful challenge,
           * no update database
           */

          IF secure server THEN
               send NEGATIVE NAME REGISTRATION RESPONSE;
          ELSE
          BEGIN /* new entry */
               IF entry already exists THEN
                    update data base - remove entry;
               update data base - add new entry;
               send POSITIVE NAME REGISTRATION RESPONSE;
               start_timer(TTL);
          END

     NAME REFRESH REQUEST:
          check for consistency;
```

```
            IF node not allowed to have name THEN
            BEGIN

                 /*
                  * tell end node that it can't have name
                  */
                 send NEGATIVE NAME REGISTRATION RESPONSE;
            END
            ELSE
            BEGIN

                 /*
                  * send confirmation response to the
                  * end node.
                  */
                 send POSITIVE NAME REGISTRATION;
                 start_timer(TTL);
            END
            return;
       END /* case */
  END /* procedure */
```

5.1.4.2. NBNS TIMER INITIATED PROCESSING

A NS node uses timers to flush out entries from the data base. Each entry in the data base is removed when its timer expires. This time value is a multiple of the refresh TTL established when the name was registered.

```
PROCEDURE timer_expired()

/*
 * processing initiated by expiration of TTL for a given name
 */

BEGIN
     /*
      * NBNS can (optionally) ensure
      * that the node is actually down
      * by sending a NODE STATUS REQUEST.
      * If such a request is sent, and
      * no response is received, it can
      * be assumed that the node is down.
      */
     remove entry from data base;
END
```

5.2. SESSION SERVICE PROTOCOLS

The following are variables and should be configurable by the NetBIOS user. The default values of these variables is found in "Defined Constants and Variables" in the Detailed Specification.):

- SSN_RETRY_COUNT - The maximum number TCP connection attempts allowable per a single NetBIOS call request.

- SSN_CLOSE_TIMEOUT is the time period to wait when closing the NetBIOS session before killing the TCP connection if session sends are outstanding.

The following are Defined Constants for the NetBIOS Session Service. (See "Defined Constants and Variables" in the Detailed Specification for the value of these constants):

- SSN_SRVC_TCP_PORT - is the globally well-known TCP port allocated for the NetBIOS Session Service. The service accepts TCP connections on this port to establish NetBIOS Sessions. The TCP connection established to this port by the caller is initially used for the exchange of NetBIOS control information. The actual NetBIOS data connection may also pass through this port or, through the retargetting facility, through another port.

5.2.1. SESSION ESTABLISHMENT PROTOCOLS

5.2.1.1. USER REQUEST PROCESSING

```
PROCEDURE listen(listening name, caller name)
/*
 * User initiated processing for B, P and M nodes
 *
 * This procedure assumes that an incoming session will be
 * retargetted here by a session server.
 */
BEGIN
     Do TCP listen; /* Returns TCP port used */
     Register listen with Session Service, give names and
          TCP port;

     Wait for TCP connection to open;   /* Incoming call */

     Read SESSION REQUEST packet from connection

     Process session request (see section on
          processing initiated by the reception of session
          service packets);
```

```
        Inform Session Service that NetBIOS listen is complete;

        IF session established THEN
             return success and session information to user;
        ELSE
             return failure;
   END /* procedure */

   PROCEDURE call(calling name, called name)
   /*
    * user initiated processing for B, P and M nodes
    */

   /*
    * This algorithm assumes that the called name is a unique name.
    * If the called name is a group name, the call() procedure
    * needs to cycle through the members of the group
    * until either (retry_count == SSN_RETRY_COUNT) or
    * the list has been exhausted.
    */
   BEGIN
        retry_count = 0;
        retarget = FALSE;    /* TRUE: caller is being retargetted */
        name_query = TRUE;   /* TRUE: caller must begin again with */
                             /*       name query. */

        REPEAT
             IF name_query THEN
             BEGIN
                  do name discovery, returns IP address;
                  TCP port = SSN_SRVC_TCP_PORT;

                  IF name discovery fails THEN
                       return failure;
                  ELSE
                       name_query = FALSE;
             END

             /*
              * now have IP address and TCP port of
              * remote party.
              */

             establish TCP connection with remote party, use an
                  ephemeral port as source TCP port;
             IF connection refused THEN
             BEGIN
                  IF retarget THEN
                  BEGIN
                       /* retry */
                       retarget = FALSE;
```

```
                use original IP address and TCP port;
                goto LOOP;
           END

           /* retry for just missed TCP listen */

           pause(SESSION_RETRY_TIMER);
           establish TCP connection, again use ephemeral
                port as source TCP port;

           IF connection refused OR
              connection timed out THEN
                return failure;
      END
      ELSE
      IF connection timed out THEN
      BEGIN
           IF retarget THEN
           BEGIN
                /* retry */
                retarget = FALSE;
                use original IP address and TCP port;
                goto LOOP;
           END
           ELSE
           BEGIN
                /*
                 * incorrect name discovery was done,
                 * try again
                 */

                inform name discovery process of
                     possible error;
                name_query = TRUE;
                goto LOOP;
           END
      END

      /*
       * TCP connection has been established
       */

      wait for session response packet;
      CASE packet type OF

         POSITIVE SESSION RESPONSE:
           return success and session established
                information;

         NEGATIVE SESSION RESPONSE:
         BEGIN
```

```
                    CASE error OF
                       NOT LISTENING ON CALLED NAME:
                       NOT LISTENING FOR CALLING NAME:
                       BEGIN
                         kill TCP connection;
                         return failure;
                       END

                       CALLED NAME NOT PRESENT:
                       BEGIN
                         /*
                          * called name does not exist on
                          * remote node
                          */

                         inform name discovery procedure
                              of possible error;

                         IF this is a P or M node THEN
                         BEGIN
                              /*
                               * Inform NetBIOS Name Server
                               * it has returned incorrect
                               * information.
                               */
                              send NAME RELEASE REQUEST for called
                                name and IP address to
                                NetBIOS Name Server;
                         END
                         /* retry from beginning */
                         retarget = FALSE;
                         name_query = TRUE;
                         goto LOOP;
                       END /* called name not present */
                    END /* case */
                  END /* negative response */

                  RETARGET SESSION RESPONSE:
                  BEGIN
                    close TCP connection;
                    extract IP address and TCP port from
                         response;
                    retarget = TRUE;
                  END /* retarget response */
               END /* case */

LOOP:          retry_count = retry_count + 1;

     UNTIL (retry_count > SSN_RETRY_COUNT);
     return failure;
END /* procedure */
```

5.2.1.2. RECEIVED PACKET PROCESSING

These are packets received on a TCP connection before a session has been established. The listen routines attached to a NetBIOS user process need not implement the RETARGET response section. The user process version, separate from a shared Session Service, need only accept (POSITIVE SESSION RESPONSE) or reject (NEGATIVE SESSION RESPONSE) a session request.

```
PROCEDURE session_packet(packet)
/*
 * processing initiated by receipt of a session service
 * packet for a session in the session establishment phase.
 * Assumes the TCP connection has been accepted.
 */
BEGIN
     CASE packet type

        SESSION REQUEST:
        BEGIN
          IF called name does not exist on node THEN
          BEGIN
               send NEGATIVE SESSION RESPONSE with CALLED
                    NAME NOT PRESENT error code;
               close TCP connection;
          END

          Search for a listen with CALLING NAME for CALLED
               NAME;
          IF matching listen is found THEN
          BEGIN
               IF port of listener process is port TCP
                  connection is on THEN
               BEGIN
                    send POSITIVE SESSION RESPONSE;

                    Hand off connection to client process
                         and/or inform user session is
                         established;
               END
               ELSE
               BEGIN
                    send RETARGET SESSION RESPONSE with
                         listener's IP address and
                         TCP port;
                    close TCP connection;
               END
          END
          ELSE
          BEGIN
               /* no matching listen pending */
```

```
                send NEGATIVE SESSION RESPONSE with either
                     NOT LISTENING ON CALLED NAME or NOT
                     LISTENING FOR CALLING NAME error
                     code;
                close TCP connection;
            END
          END /* session request */
       END /* case */
  END /* procedure */
```

5.2.2. SESSION DATA TRANSFER PROTOCOLS

5.2.2.1. USER REQUEST PROCESSING

```
  PROCEDURE send_message(user_message)
  BEGIN
       build SESSION MESSAGE header;
       send SESSION MESSAGE header;
       send user_message;
       reset and restart keep-alive timer;
       IF send fails THEN
       BEGIN
            /*
             * TCP connection has failed */
             */
            close NetBIOS session;
            inform user that session is lost;
            return failure;
       END
       ELSE
            return success;
  END
```

5.2.2.2. RECEIVED PACKET PROCESSING

These are packets received after a session has been established.

```
  PROCEDURE session_packet(packet)
  /*
   * processing initiated by receipt of a session service
   * packet for a session in the data transfer phase.
   */
  BEGIN
       CASE packet type OF

          SESSION  MESSAGE:
          BEGIN
            process message header;
            read in user data;
            reset and restart keep-alive timer;
            deliver data to user;
```

```
            END /* session message */

            SESSION KEEP ALIVE:
              discard packet;

         END /* case */
    END /* procedure */
```

5.2.2.3. PROCESSING INITIATED BY TIMER

```
    PROCEDURE session_ka_timer()
    /*
     * processing initiated when session keep alive timer expires
     */
    BEGIN
             send SESSION KEEP ALIVE, if configured;
             IF send fails THEN
             BEGIN
              /* remote node, or path to it, is down */

              abort TCP connection;
              close NetBIOS session;
              inform user that session is lost;
              return;
             END
    END /* procedure */
```

5.2.3. SESSION TERMINATION PROTOCOLS

5.2.3.1. USER REQUEST PROCESSING

```
    PROCEDURE close_session()

    /* initiated by a user request to close a session */

    BEGIN
         close gracefully the TCP connection;

         WAIT for the connection to close or SSN_CLOSE_TIMEOUT
              to expire;

         IF time out expired THEN
              abort TCP connection;
    END /* procedure */
```

5.2.3.2. RECEPTION INDICATION PROCESSING

```
    PROCEDURE close_indication()
    /*
     * initiated by a TCP indication of a close request from
     * the remote connection partner.
```

```
 */
BEGIN
     close gracefully TCP connection;

     close NetBIOS session;

     inform user session closed by remote partner;
END /* procedure */
```

5.3. NetBIOS DATAGRAM SERVICE PROTOCOLS

The following are GLOBAL variables and should be NetBIOS user configurable:

- SCOPE_ID: the non-leaf section of the domain name preceded by a '.' which represents the domain of the NetBIOS scope for the NetBIOS name. The following protocol description only supports single scope operation.

- MAX_DATAGRAM_LENGTH: the maximum length of an IP datagram. The minimal maximum length defined in for IP is 576 bytes. This value is used when determining whether to fragment a NetBIOS datagram. Implementations are expected to be capable of receiving unfragmented NetBIOS datagrams up to their maximum size.

- BROADCAST_ADDRESS: the IP address B-nodes use to send datagrams with group name destinations and broadcast datagrams. The default is the IP broadcast address for a single IP network.

The following are Defined Constants for the NetBIOS Datagram Service:

- DGM_SRVC_UDP_PORT: the globally well-known UDP port allocated where the NetBIOS Datagram Service receives UDP packets. See section 6, "Defined Constants", for its value.

5.3.1. B NODE TRANSMISSION OF NetBIOS DATAGRAMS

```
PROCEDURE send_datagram(data, source, destination, broadcast)

/*
 * user initiated processing on B node
 */

BEGIN
     group = FALSE;

     do name discovery on destination name, returns name type and
          IP address;
```

```
IF name type is group name THEN
BEGIN
     group = TRUE;
END

/*
 * build datagram service UDP packet;
 */
convert source and destination NetBIOS names into
     half-ASCII, biased encoded name;
SOURCE_NAME = cat(source, SCOPE_ID);
SOURCE_IP = this nodes IP address;
SOURCE_PORT =  DGM_SRVC_UDP_PORT;

IF NetBIOS broadcast THEN
BEGIN
     DESTINATION_NAME = cat("*", SCOPE_ID)
END
ELSE
BEGIN
     DESTINATION_NAME = cat(destination, SCOPE_ID)
END

MSG_TYPE = select_one_from_set
     {BROADCAST, DIRECT_UNIQUE, DIRECT_GROUP}
DGM_ID = next transaction id for Datagrams;
DGM_LENGTH = length of data + length of second level encoded
     source and destination names;

IF (length of the NetBIOS Datagram, including UDP and
    IP headers, > MAX_DATAGRAM_LENGTH) THEN
BEGIN
     /*
      * fragment NetBIOS datagram into 2 UDP packets
      */
     Put names into 1st UDP packet and any data that fits
          after names;
     Set MORE and FIRST bits in 1st UDP packet's FLAGS;
     OFFSET in 1st UDP = 0;

     Replicate NetBIOS Datagram header from 1st UDP packet
          into 2nd UDP packet;
     Put rest of data in 2nd UDP packet;
     Clear MORE and FIRST bits in 2nd UDP packet's FLAGS;
     OFFSET in 2nd UDP = DGM_LENGTH - number of name and
          data bytes in 1st UDP;
END
BEGIN
     /*
      * Only need one UDP packet
      */
```

```
            USER_DATA = data;
            Clear MORE bit and set FIRST bit in FLAGS;
            OFFSET = 0;
       END

       IF (group == TRUE) OR (NetBIOS broadcast) THEN
       BEGIN
            send UDP packet(s) to BROADCAST_ADDRESS;
       END
       ELSE
       BEGIN
            send UDP packet(s) to IP address returned by name
               discovery;
       END
  END /* procedure */
```

5.3.2. P AND M NODE TRANSMISSION OF NetBIOS DATAGRAMS

```
  PROCEDURE send_datagram(data, source, destination, broadcast)

  /*
   * User initiated processing on P and M node.
   *
   * This processing is the same as for B nodes except for
   * sending broadcast and multicast NetBIOS datagrams.
   */

  BEGIN
       group = FALSE;

       do name discovery on destination name, returns name type
            and IP address;
       IF name type is group name THEN
       BEGIN
            group = TRUE;
       END

       /*
        * build datagram service UDP packet;
        */
       convert source and destination NetBIOS names into
            half-ASCII, biased encoded name;
       SOURCE_NAME = cat(source, SCOPE_ID);
       SOURCE_IP = this nodes IP address;
       SOURCE_PORT =  DGM_SRVC_UDP_PORT;

       IF NetBIOS broadcast THEN
       BEGIN
            DESTINATION_NAME = cat("*", SCOPE_ID)
       END
       ELSE
```

```
BEGIN
     DESTINATION_NAME = cat(destination, SCOPE_ID)
END

MSG_TYPE = select_one_from_set
     {BROADCAST, DIRECT_UNIQUE, DIRECT_GROUP}
DGM_ID = next transaction id for Datagrams;
DGM_LENGTH = length of data + length of second level encoded
     source and destination names;

IF (length of the NetBIOS Datagram, including UDP and
    IP headers, > MAX_DATAGRAM_LENGTH) THEN
BEGIN
     /*
      * fragment NetBIOS datagram into 2 UDP packets
      */
     Put names into 1st UDP packet and any data that fits
          after names;
     Set MORE and FIRST bits in 1st UDP packet's FLAGS;

     OFFSET in 1st UDP = 0;

     Replicate NetBIOS Datagram header from 1st UDP packet
          into 2nd UDP packet;
     Put rest of data in 2nd UDP packet;
     Clear MORE and FIRST bits in 2nd UDP packet's FLAGS;
     OFFSET in 2nd UDP = DGM_LENGTH - number of name and
          data bytes in 1st UDP;
END
BEGIN
     /*
      * Only need one UDP packet
      */
     USER_DATA = data;
     Clear MORE bit and set FIRST bit in FLAGS;
     OFFSET = 0;
END

IF (group == TRUE) OR (NetBIOS broadcast) THEN
BEGIN
     /*
      * Sending of following query is optional.
      * Node may send datagram to NBDD immediately
      * but NBDD may discard the datagram.
      */
     send DATAGRAM QUERY REQUEST to NBDD;
     IF response is POSITIVE QUERY RESPONSE THEN
          send UDP packet(s) to NBDD Server IP address;
     ELSE
     BEGIN
          get list of destination nodes from NBNS;
```

```
                FOR EACH node in list
                BEGIN
                     send UDP packet(s) to this node's
                          IP address;
                END
           END
      END
      ELSE
      BEGIN
           send UDP packet(s) to IP address returned by name
              discovery;
END /* procedure */
```

5.3.3. RECEPTION OF NetBIOS DATAGRAMS BY ALL NODES

The following algorithm discards out of order NetBIOS Datagram fragments. An implementation which reassembles out of order NetBIOS Datagram fragments conforms to this specification. The fragment discard timer is initialized to the value FRAGMENT_TO. This value should be user configurable. The default value is given in Section 6, "Defined Constants and Variables".

```
PROCEDURE datagram_packet(packet)

/*
 * processing initiated by datagram packet reception
 * on B, P and M nodes
 */
BEGIN
     /*
      * if this node is a P node, ignore
      * broadcast packets.
      */

     IF this is a P node AND incoming packet is
          a broadcast packet THEN
     BEGIN
          discard packet;
     END

     CASE packet type OF

        DATAGRAM SERVICE:
        BEGIN
          IF FIRST bit in FLAGS is set THEN
          BEGIN
               IF MORE bit in FLAGS is set THEN
               BEGIN
                    Save 1st UDP packet of the Datagram;
                    Set this Datagram's fragment discard
                      timer to FRAGMENT_TO;
```

```
                return;
           END
           ELSE
                Datagram is composed of a single
                  UDP packet;
      END
      ELSE
      BEGIN
           /* Have the second fragment of a Datagram */

           Search for 1st fragment by source IP address
              and DGM_ID;
           IF found 1st fragment THEN
                Process both UDP packets;
           ELSE
           BEGIN
                discard 2nd fragment UDP packet;
                return;
           END
      END

      IF DESTINATION_NAME is '*' THEN
      BEGIN
           /* NetBIOS broadcast */

           deliver USER_DATA from UDP packet(s) to all
                outstanding receive broadcast
                datagram requests;
           return;
      END
      ELSE
      BEGIN /* non-broadcast */
           /* Datagram for Unique or Group Name */

           IF DESTINATION_NAME is not present in the
              local name table THEN
           BEGIN
                /* destination not present */
                build DATAGRAM ERROR packet, clear
                     FIRST and MORE bit, put in
                     this nodes IP and PORT, set
                     ERROR_CODE;
                send DATAGRAM ERROR packet to
                     source IP address and port
                     of UDP;
                discard UDP packet(s);
                return;
           END
           ELSE
           BEGIN /* good */
                /*
```

```
                        * Replicate received NetBIOS datagram for
                        * each recipient
                        */
                       FOR EACH pending NetBIOS user's receive
                            datagram operation
                       BEGIN
                            IF source name of operation
                               matches destination name
                               of packet THEN
                            BEGIN
                               deliver USER_DATA from UDP
                                 packet(s);
                            END
                       END /* for each */
                       return;
                    END /* good */
                 END /* non-broadcast */
              END /* datagram service */

            DATAGRAM ERROR:
            BEGIN
                 /*
                  * name service returned incorrect information
                  */

                 inform local name service that incorrect
                   information was provided;

                 IF this is a P or M node THEN
                 BEGIN
                      /*
                       * tell NetBIOS Name Server that it may
                       * have given incorrect information
                       */

                      send NAME RELEASE REQUEST with name
                        and incorrect IP address to NetBIOS
                        Name Server;
                 END
            END /* datagram error */

       END /* case */
  END
```

5.3.4. PROTOCOLS FOR THE NBDD

The key to NetBIOS Datagram forwarding service is the packet delivered to the destination end node must have the same NetBIOS header as if the source end node sent the packet directly to the destination end node. Consequently, the NBDD does not reassemble NetBIOS Datagrams. It forwards the UDP packet as is.

```
PROCEDURE  datagram_packet(packet)

/*
 * processing initiated by a incoming datagram service
 * packet on a NBDD node.
 */

BEGIN
     CASE packet type OF

        DATAGRAM SERVICE:
        BEGIN
             IF packet was sent as a directed
                NetBIOS datagram THEN
             BEGIN
               /*
                * provide group forwarding service
                *
                * Forward datagram to each member of the
                * group.  Can forward via:
                *   1) get list of group members and send
                *   the DATAGRAM SERVICE packet unicast
                *   to each
                *   2) use Group Multicast, if available
                *   3) combination of 1) and 2)
                */

               ...

             END

             ELSE
             BEGIN
               /*
                * provide broadcast forwarding service
                *
                * Forward datagram to every node in the
                * NetBIOS scope.  Can forward via:
                *   1) get list of group members and send
                *   the DATAGRAM SERVICE packet unicast
                *   to each
                *   2) use Group Multicast, if available
                *   3) combination of 1) and 2)
                */

               ...

             END
        END /* datagram service */

        DATAGRAM ERROR:
```

```
          BEGIN
            /*
             * Should never receive these because Datagrams
             * forwarded have source end node IP address and
             * port in NetBIOS header.
             */

            send DELETE NAME REQUEST with incorrect name and
                 IP address to NetBIOS Name Server;

          END /* datagram error */

          DATAGRAM QUERY REQUEST:
          BEGIN
            IF can send packet to DESTINATION_NAME THEN
            BEGIN
                 /*
                  * NBDD is able to relay Datagrams for
                  * this name
                  */

                 send POSITIVE DATAGRAM QUERY RESPONSE to
                   REQUEST source IP address and UDP port
                   with request's DGM_ID;
            END
            ELSE
            BEGIN
                 /*
                  * NBDD is NOT able to relay Datagrams for
                  * this name
                  */

                 send NEGATIVE DATAGRAM QUERY RESPONSE to
                   REQUEST source IP address and UDP port

                   with request's DGM_ID;
            END
          END /* datagram query request */

     END /* case */
END /* procedure */
```

6. DEFINED CONSTANTS AND VARIABLES

GENERAL:

SCOPE_ID	The name of the NetBIOS scope. This is expressed as a character string meeting the requirements of the domain name system and without a leading or trailing "dot". An implementation may elect to make this a single global value for the node or allow it to be specified with each separate NetBIOS name (thus permitting cross-scope references.)
BROADCAST_ADDRESS	An IP address composed of the nodes's network and subnetwork numbers with all remaining bits set to one. I.e. "Specific subnet" broadcast addressing according to section 2.3 of RFC 950.
BCAST_REQ_RETRY_TIMEOUT	250 milliseconds. An adaptive timer may be used.
BCAST_REQ_RETRY_COUNT	3
UCAST_REQ_RETRY_TIMEOUT	5 seconds An adaptive timer may be used.
UCAST_REQ_RETRY_COUNT	3
MAX_DATAGRAM_LENGTH	576 bytes (default)

NAME SERVICE:

REFRESH_TIMER	Negotiated with NBNS for each name.
CONFLICT_TIMER	1 second Implementations may chose a longer value.
NAME_SERVICE_TCP_PORT	137 (decimal)

NAME_SERVICE_UDP_PORT	137 (decimal)
INFINITE_TTL	0

SESSION SERVICE:

SSN_SRVC_TCP_PORT	139 (decimal)
SSN_RETRY_COUNT	4 (default) Re-configurable by user.
SSN_CLOSE_TIMEOUT	30 seconds (default) Re-configurable by user.
SSN_KEEP_ALIVE_TIMEOUT	60 seconds, recommended, may be set to a higher value. (Session keep-alives are used only if configured.)

DATAGRAM SERVICE:

DGM_SRVC_UDP_PORT	138 (decimal)
FRAGMENT_TO	2 seconds (default)

REFERENCES

[1] "Protocol Standard For a NetBIOS Service on a TCP/UDP Transport: Concepts and Methods", RFC 1001, March 1987.

[2] J. Reynolds, J. Postel, "Assigned Numbers", RFC 990, November 1986.

[3] P. Mockapetris, "Domain Names - Implementation and Specification", RFC 883, November 1983.

Glossary

ACL
(Access Control List) A list used to control access to a file or resource. The list contains the user IDs and/or group IDs that are allowed access to the file or resource.

API
(Application Programming Interface) A published interface for software developers.

big-endian
The name of a particular byte order (coined by Danny Cohen). When looking at addresses in increasing order, the most significant byte comes first. The Internet protocols use big-endian byte order.

broadcast
The function of delivering a given packet to all hosts that are attached to the broadcasting delivery system. Broadcasting is implemented both at the hardware and the software levels.

byte
8bits.

CAE
Common Applications Environment.

chaining
Transmission of more than one SMB request in a request.

client-server
The distributed system model where a requesting program (the client) interacts with a program that can satisfy the request (the server). The client initiates the interaction and may wait for the server to respond.

connection-oriented service
A service provided between two endpoints along which data is passed in a sequenced and reliable way.

connectionless service
In a connectionless service each packet is a separate entity containing a source and destination address; therefore, packets may be dropped or delivered out of sequence. The delivery service offered by the Internet Protocol (IP) is a connectionless service.

core
The dialect name for the basic SMB dialect described in this specification.

core plus
The dialect name for the SMB dialect that provides additional features to the core dialect.

data encapsulation
The way a lower-level protocol accepts a message from a higher-level protocol and places it in the data portion of the low-level frame.

daemon
A process that is not associated with any user. This sort of process performs system-wide functions; for example, administration, control of networks and execution-dependent activities.

datagram
A packet sent independently of the others in the network. It contains the source and destination addresses as well as the data.

dialect
Used to refer to the level of protocol negotiated between the SMB redirector and the LMX server.

DES
U.S. Department of Commerce Data Encryption Standard.

EA
(Extended Attribute) An SMB protocol element supported by the extended 2.0 protocol dialect. Extended attributes can be associated with a file.

effective group ID
An attribute of a process that is used in determining various permissions, including file access permissions. This value is subject to change during the process' lifetime.

effective user ID
An attribute of a process that is used in determining various permissions, including file access permissions. This value is subject to change during the process' lifetime.

exec
The XSI system call that is used to start a process running.

extended 1.0
The dialect name for the first extended SMB protocol dialect.

extended 2.0
The dialect name for the second extended SMB protocol dialect.

Extended Attribute
See EA.

FCB
(File Control Block) The area of memory holding the file information and status. It is a term associated with DOS.

FID
(File ID) A unique number associated with a file to enable it to be identified.

fifo
(First In First Out) One of the file types supported on an XSI system. A fifo, the alternative name for a pipe, differs from a regular file because its data is transient; that is, once data is read from the pipe it cannot be read again.

fork
The XSI system call which is used to create a new process. The process created is a duplicate of the calling process.

Internet Protocol
(IP) The protocol from the Internet Protocol Suite that provides the basis for Internet communications.

interoperability
The ability of software and hardware on multiple machines and from multiple vendors to communicate effectively.

ioctl
A system call which allows a process to specify control information to control a device. This

function exists in both XSI and DOS.

IPC
(Inter-process Communication) Methods by which two or more processes can communicate; for example, formatted data streams or shared memory.

LAN
(Local Area Network) A physical network that operates at a high speed over short distances; for example, Ethernet.

little-endian
The name of a particular byte order (coined by Danny Cohen). When looking at addresses in increasing order, the least significant byte comes first.

LMX
X/Open LAN Manager Architecture. The implementation of the LAN Manager on CAE systems.

LMX Server
The system providing the LMX service.

LMX Session
The path between two communicating systems that provides a reliable, sequenced data delivery service.

MBZ
(Must Be Zero) Reserved fields are often defined MBZ.

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

multicast
A method by which copies of a single packet are passed to a selected subset of all destinations. Broadcast is a special case of multicast whereby the subset of destinations receiving a copy of the packet is the entire set of destinations.

named pipe
An inter-process communication mechanism defined by the extended SMB specification. Also a fifo.

NetBIOS
(Network Basic Input Output System) The *de facto* standard programmatic interface to networks for DOS systems.

NFS
(Network File System) A protocol which allows a set of computers access to each others' file systems. NFS was developed by Sun Microsystems and is used primarily on UNIX systems.

octet
8 bits.

opportunistic lock
The server will notify the client, allowing it to flush its dirty buffers and unlock the file, when another client attempts to open the file.

OSI
(Open Systems Interconnect) ISO standards for the interconnection of cooperative (open) computer systems.

packet
A block of data sent across a packet switching network.

PID
(Process ID) The number assigned to a process so that it can be uniquely identified.

responder
An entity with which an initiator wishes to establish a transport connection.

RFC
(Request for Comments) The name of a series of notes that contain surveys, measurements, ideas, techniques and observations, as well as proposed and accepted Internet protocol standards.

root (of file system)
The top directory in the directory hierarchical structure.

RPC
Remote Procedure Call.

session
See LMX Session.

SMB
(Server Message Block) A protocol which allows a set of computers to access shared resources as if they were local. The core protocol was developed by Microsoft Corporation and Intel, and the extended protocols were developed by Microsoft Corporation.

SMB redirector
The client system accessing the LMX server.

SMB request
The server message block sent from the SMB redirector to the LMX server.

SMB response
The server message block sent from the LMX server to the SMB redirector.

TBD
(To be Defined) Further detail will be provided at a later time.

TCP
(Transmission Control Protocol) The Internet standard transport level connection-oriented protocol. It provides a full duplex, reliable stream service which allows a process on one machine to send a stream of data to a process on another. Part of the Internet Protocol Suite.

TID
(Tree Connect Identifier) A numeric value passed by the LMX server to the SMB redirector to represent a location within a file system subtree.

UDP
(User Datagram Protocol) The Internet connectionless protocol. Part of the Internet Protocol Suite.

UID
(User Identifier) A token representing an authenticated *<username, password>* tuple. UIDs are registered by the redirectors.

umask
The XSI process' file mode creation mask used during file and directory creation. Bit positions that are set in the umask are cleared in the mode of the newly created file or directory. The

umask is set using the *umask*() call.

working directory
A directory, associated with a process, that is used in pathname resolution for pathnames that do not begin with a slash.

Index