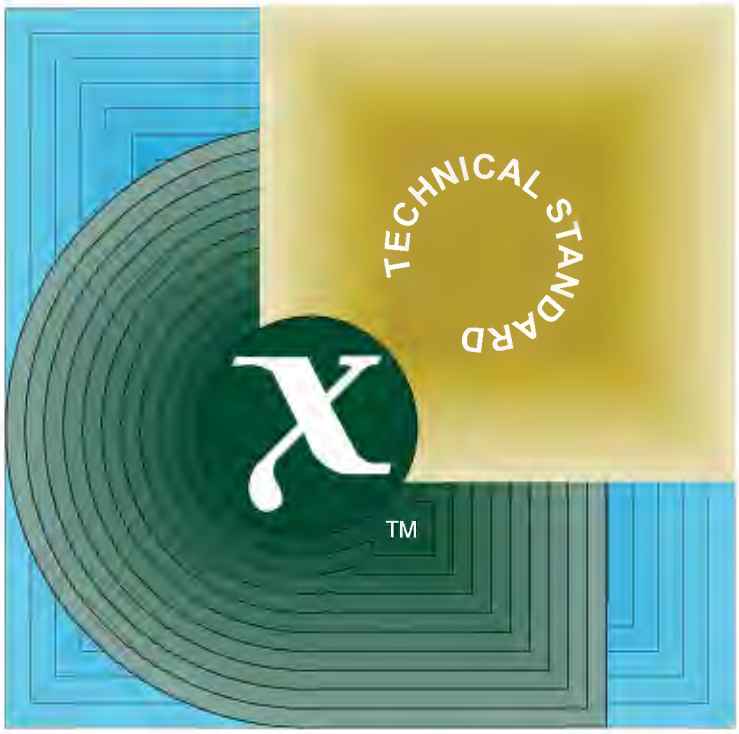
Technical Standard

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: 1 872630 45 6

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

Chapter	**1**	**Introduction**	**1**
	1.1	Why Republish	1
	1.2	This Document	1
	1.3	Overview of Document Layout	2
Chapter	**2**	**SMB File-sharing Service Model**	**3**
	2.1	SMB Protocol Principles	4
	2.2	Security Overview	5
	2.2.1	Share-level Security Mode	5
	2.2.2	User-level Security Mode	5
Chapter	**3**	**SMB Protocol Conventions**	**7**
	3.1	Summary of SMBs	7
	3.2	SMB Environment Definitions	10
	3.3	Share-level and User-level Security Modes	12
	3.3.1	Share-level Security Mode	12
	3.3.2	User-level Security Mode with Extended Protocols	12
	3.3.3	User-level Security with Core Protocol	13
	3.4	Connection Protocols	14
	3.5	Naming	15
	3.5.1	Resource Names	15
	3.5.2	NetBIOS Names	15
	3.5.3	Uniform Naming Convention	16
	3.5.4	Canonical Pathnames	16
	3.5.5	Long Names	16
	3.6	Wildcards	17
	3.7	File Paradigm	17
	3.7.1	Regular Files	18
	3.7.2	Open Modes	18
	3.7.3	Write Behaviour	19
	3.8	Locking Conventions	20
	3.8.1	Byte Locking	20
	3.8.2	Opportunistic Locking	20
	3.9	Chaining of Extended SMB Requests	22
	3.10	Exception and Error Handling	24
	3.10.1	Disorderly LMX Session Dissolution	24
	3.10.2	Errors and Error Handling	24
	3.11	Timeouts	25
	3.12	Downward-compatibility Support	25

Chapter	**4**	**LMX Considerations**	**27**
	4.1	LMX Username Mapping	27
	4.2	LMX Filename Mapping	28
	4.3	LMX File Mapping	30
	4.3.1	SMB File Attributes	30
	4.3.2	CAE File Access Permissions	30
	4.3.3	File System Issues	30
	4.3.4	CAE Special Files	31
	4.3.5	Deleting or Renaming a File	31
	4.3.6	Long Filenames	31
	4.3.7	Extended Attributes	31
	4.4	LMX File Locking	33
	4.4.1	Interlocking Behaviour	33
	4.4.2	Locking Timeouts	34
	4.4.3	Read-only Locks	34
	4.5	LMX Server Caching	35
	4.6	LMX Print Spooling	35
	4.7	SMB Error Codes	35
	4.8	Security Policy	36
	4.9	Negotiated Dialect	36
	4.10	Network Issues	36
Chapter	**5**	**Data Objects and Constants**	**37**
	5.1	SMB Format	37
	5.2	SMB Command Codes	40
	5.3	Data Objects	43
	5.3.1	Time Fields	43
	5.3.2	Date Fields	43
	5.3.3	File Attributes Fields	43
	5.3.4	Buffers	44
	5.3.5	File-sharing Control	44
	5.3.6	Resource Types	45
	5.3.7	Access Modes	46
	5.3.8	Open Function	46
	5.3.9	Resource Names, Pathnames, Filenames and Network Pathnames	46
	5.3.10	File Identifiers	47
	5.4	SMB Dialects	48
	5.5	Timeouts	48
	5.6	SMB Error Codes	49
	5.6.1	SMB Error Class Mappings	49
	5.6.2	Error Codes for the SUCCESS Class	49
	5.6.3	Error Codes for the ERRDOS Class	49
	5.6.4	Error Codes for the ERRSRV Class	51
	5.6.5	Error Codes for the ERRHRD Class	52

Chapter 6 Core SMB Connection Management Requests 55
6.1 SMBnegprot Specification 55
6.2 SMBtcon Specification 57
6.3 SMBtdis Specification 59
6.4 SMBexit Specification 61

Chapter 7 Core SMB File Operation Requests 63
7.1 SMBcreate Specification 63
7.2 SMBmknew Specification 67
7.3 SMBopen Specification 70
7.4 SMBread Specification 73
7.5 SMBwrite Specification 76
7.6 SMBlseek Specification 79
7.7 SMBlock Specification 81
7.8 SMBunlock Specification 83
7.9 SMBflush Specification 85
7.10 SMBclose Specification 87
7.11 SMBmv Specification 89
7.12 SMBunlink Specification 92

Chapter 8 Core SMB Directory and Attribute Operations 95
8.1 SMBmkdir Specification 95
8.2 SMBrmdir Specification 97
8.3 SMBsearch Specification 99
8.4 SMBgetatr Specification 103
8.5 SMBsetatr Specification 105
8.6 SMBdskattr Specification 107
8.7 SMBchkpath Specification 109

Chapter 9 Core SMB Spool Operation Requests 111
9.1 SMBsplopen Specification 111
9.2 SMBsplwr Specification 113
9.3 SMBsplclose Specification 115
9.4 SMBsplretq Specification 117

Chapter 10 Core Plus SMB File Operations 121
10.1 SMBnegprot Specification 121
10.2 SMBreadbraw Specification 123
10.3 SMBwritebraw Specification 125
10.4 SMBlockread Specification 128
10.5 SMBwriteunlock Specification 130
10.6 SMBwriteclose Specification 132

Chapter 11 Extended 1.0 SMB Connection Management Requests 135
11.1 SMBnegprot Specification 135
11.2 SMBsecpkgX Specification 139
11.3 SMBsesssetupX Specification 144
11.4 SMBtconX Specification 147

Chapter 12 Extended 1.0 SMB File Operations 151
12.1 SMBopenX Specification 151
12.2 SMBlockingX Specification 156
12.3 SMBreadX Specification 160
12.4 SMBwritebraw Specification 163
12.5 SMBwriteclose Specification 166
12.6 SMBwriteX Specification 168
12.7 SMBreadbmpx Specification 171
12.8 SMBwritebmpx Specification 174

Chapter 13 Extended 1.0 SMB Directory and Attribute Operations 179
13.1 SMBffirst Specification 179
13.2 SMBfclose Specification 181
13.3 SMBfunique Specification 182
13.4 SMBgetattrE Specification 183
13.5 SMBsetattrE Specification 185

Chapter 14 Extended 1.0 SMB Miscellaneous Requests 187
14.1 SMBcopy Specification 187
14.2 SMBecho Specification 191
14.3 SMBioctl Specification 193
14.4 SMBmove Specification 194

Chapter 15 Extended 2.0 Protocol Additions and Modifications 197
15.1 SMBsesssetupX Specification 197
15.2 SMBcopy Specification 201
15.3 SMBfindnclose Specification 202
15.4 SMBfindclose Specification 203
15.5 SMBuloggoffX Specification 204

Chapter 16 Extended 2.0 Protocol SMBtrans2 207
16.1 SMBtrans2 207
16.1.1 Request Formats 207
16.1.2 Response Format 209
16.1.3 Transaction Flow 210
16.1.4 Service 211
16.1.5 Extended Attribute 212
16.1.5.1 Errors Encountered When Creating EAs 212
16.1.5.2 Encapsulation of EAs in the SMB Protocol 212
16.1.5.3 FEA Structure 212
16.1.5.4 GEA Structure 214
16.1.6 Information Levels 214
16.1.7 Defined SMBtrans2 Protocols 214
16.2 TRANSACT2_OPEN 216
16.3 TRANSACT2_FINDFIRST 221
16.4 TRANSACT2_FINDNEXT 225
16.5 TRANSACT2_QFSINFO 229
16.6 TRANSACT2_SETFSINFO 231

16.7 TRANSACT2_QPATHINFO 233
16.8 TRANSACT2_SETPATHINFO 236
16.9 TRANSACT2_QFILEINFO 238
16.10 TRANSACT2_SETFILEINFO 241
16.11 TRANSACT2_FINDNOTIFYFIRST 243
16.12 TRANSACT2_FINDNOTIFYNEXT 246
16.13 TRANSACT2_MKDIR 249

Appendix A SMB Transmission Analysis 251
A.1 Introduction 251
A.2 DOS Functions 252
A.3 OS/2 Functions 259

Appendix B LAN Manager Remote Administration Protocol 263
B.1 Overview 263
B.2 Remote API Protocol 264
B.3 LMX Access Control Lists Mapping 265
B.4 Transaction API Request Format 267
B.4.1 Parameter Section 267
B.4.2 Data Section 267
B.5 Transaction API Response Format 268
B.5.1 Parameter Section 268
B.5.2 Data Section 268
B.6 Descriptor Strings 269
B.6.1 Descriptor String Types 269
B.6.2 Pointer Types and Returned Data 271
B.7 Examples 272
B.7.1 NetShareDel 272
B.7.2 NetShareAdd 272
B.7.3 NetShareEnum 273
B.8 API Numbers 275

Appendix C The X/Open Security Package 277
C.1 E() Functions 277
C.2 U() Functions 278

Appendix D SMB Encryption Techniques 279
D.1 SMB Authentication 279
D.1.1 SMBnegprot Response 279
D.1.2 SMBtcon, SMBtconX, SMBsesssetupX Requests 279

Appendix E TOP/NetBIOS 281

Appendix F RFC 1001 349

Appendix G RFC 1002 419

Glossary 505

Index 511

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.

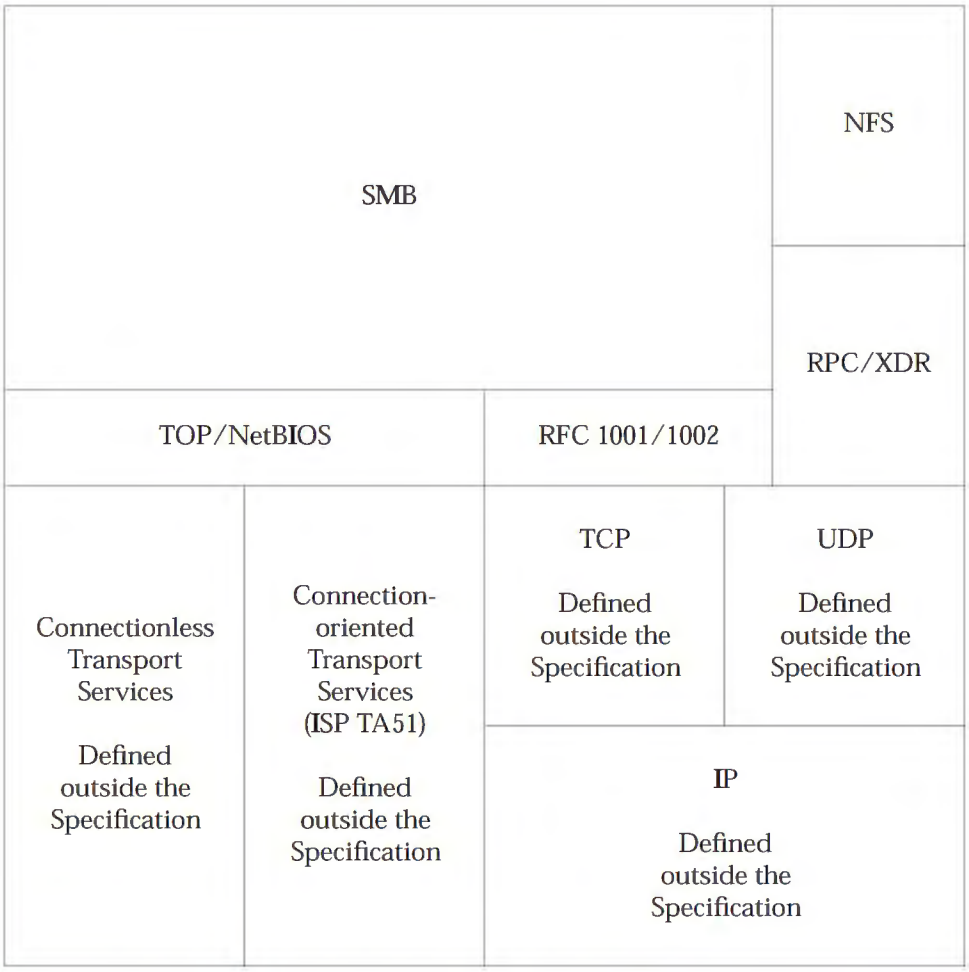
For interoperability via asynchronous serial links, X/Open has already defined in the X/Open Portability Guide, Issue 3 a file transfer protocol and a set of features provided on X/Open-compliant systems for terminal emulators. Now it is time to address interworking in local area networks (LANs).

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 (RFC1001/RFC1002).



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:

IPC
X/Open CAE Specification, IPC Mechanisms for SMB
(Document No.: C195, 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.: C196, 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.: D110, 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*	E2
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 *SMBsessetupX* request. If no *SMBsessetupX* 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:

file and pathname
resource name
LMX servername

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-PCIG, 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 A1B2.C to X1Y2.TXT, 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		New open requesting			
Deny Mode	access	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 0x00ff (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 attribute 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	
SMBsessetupX	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 3.7.2 on 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.

LPT1: 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:

\\<LMX servername>\<pathname>\<filename>

where:

<LMX servername> 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 55) 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 2.3, 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 *SMBnegprot*s 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.)
ERRrmuns	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.

Conventions

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	ERRinvnid	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

SMBmknew Detailed Description

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

SMBmknew 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.

SMBmknew 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	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.
-	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.

SMBmknew Side Effects

None.

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).

7.3 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	ERRinvnid	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 8.3 component 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 3.6 on 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 ssumed, 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	ERRinvnid	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	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.

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.

SMBslpretq 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.

Conventions

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 *SMBsessetupX* 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 *SMBsessetupX* 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 SMBsessetupX Specification

SMBsessetupX Detailed Description

This extended protocol request is used to further set up the LMX session normally just established via the *SMBnegprot* request/response. The *SMBsessetupX* 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 *SMBsessetupX* 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 *SMBsessetupX* exchange. This renegotiation is available through the *SMBtcon* request, but not through *SMBtconX*.

SMBsessetupX 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).

SMBsessetupX 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.

SMBsessetupX Preconditions

1. The SMB redirector attempting the *SMBsessetupX* 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.

SMBsessetupX 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.

SMBsessetupX 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 *SMBsessetupX* 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 5.3.6 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.

Conventions

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, SMBblockingX, SMBclose, SMBblockread* 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.

Conventions

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.

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).

13.2 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	ERRinvnid	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 SMBsessetupX Specification

SMBsessetupX 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 *SMBsessetupX* 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 *SMBsessetupX* 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 *SMBsessetupX* exchange. This renegotiation is available through the *SMBtcon* request, but not through *SMBtconX*.

SMBsessetupX Deviations

None.

SMBsessetupX Field Descriptions

From SMB redirector		To SMB redirector	
Field Name	Field Value	Field Name	Field Value
smb_com	*SMBsessetupX*	*smb_com*	*SMBsessetupX*
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 *SMBsessetupX* 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 *SMBsessetupX* 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 *SMBsessetupX* 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, *SMBsessetupX* 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 *SMBsessetupX* 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 retuned.
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 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 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.