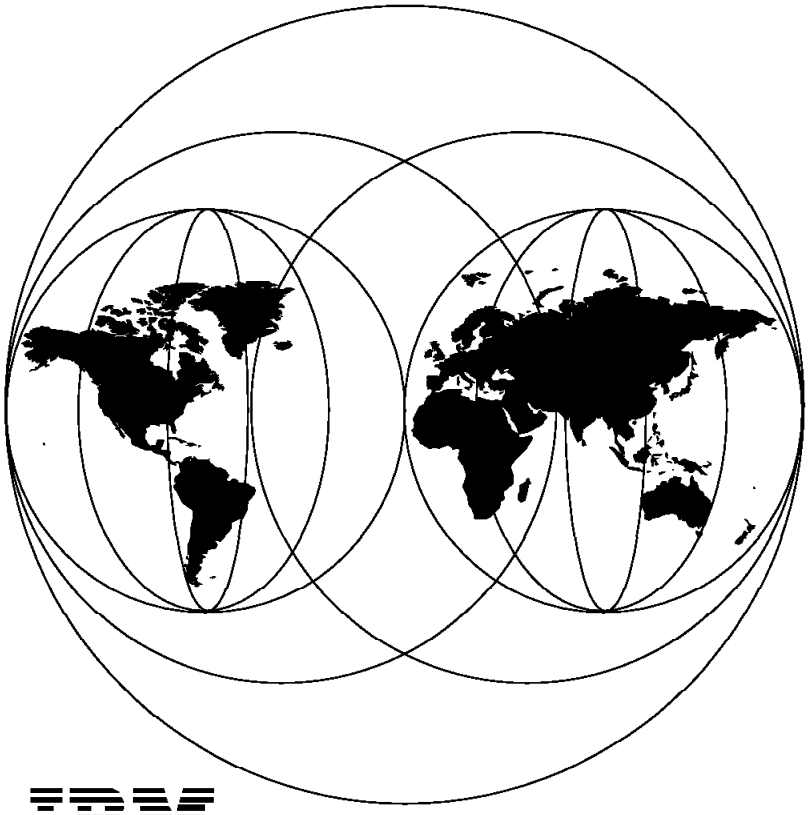
SG24-4747-00

OpenEdition for VM/ESA
Implementation and Administration Guide

August 1996



IBM
International Technical Support Organization
Poughkeepsie Center

International Technical Support Organization

SG24-4747-00

OpenEdition for VM/ESA
Implementation and Administration Guide

August 1996

Take Note!

Before using this information and the product it supports, be sure to read the general information in Appendix C, “Special Notices” on page 201.

First Edition (August 1996)

This edition applies to the OpenEdition for VM/ESA Shell and Utilities Feature and OpenEdition POSIX implementation of Virtual Machine/Enterprise Systems Architecture, Version 2 Release 1.0, Program Number 5654-030, and subsequent releases.

Comments may be addressed to:
IBM Corporation, International Technical Support Organization
Dept. HYJ Mail Station P099
522 South Road
Poughkeepsie, New York 12601-5400

When you send information to IBM, you grant IBM a non-exclusive right to use or distribute the information in any way it believes appropriate without incurring any obligation to you.

Contents

Figures ix

Tables xi

Preface xiii
How This Redbook Is Organized xiii
The Team That Wrote This Redbook xiv
Comments Welcome xv

Chapter 1. VM/ESA′s Launch into the UNIX World 1
1.1 Application Portability 1
1.2 Interoperability 2
1.3 Scalability 2
1.4 VM/ESA: The System of Choice 2

Chapter 2. OpenEdition Primer 5
2.1 A Short UNIX History Lesson 5
2.2 OpenEdition Overview 6
2.3 Introduction to the Byte File System 7
2.3.1 Terminology 7
2.3.2 Files and Directories 10
2.4 Understanding File System Permissions 11
2.4.1 Permission Bits 12
2.4.2 Directory Permission Bits 14
2.4.3 Advanced Information on Permission Bits 15
2.4.4 Default Permissions 17
2.5 Introduction to the OpenEdition Shell and Utilities 17
2.5.1 Starting the Shell 18
2.5.2 ls Shell Command 18
2.5.3 File Time Stamps 20
2.5.4 Shell Command Processing 21
2.5.5 Case Sensitivity Within the Shell 21
2.5.6 Running Programs in the Shell 22
2.5.7 Using the Shell History File 24
2.6 Help Is Always at Hand 26
2.7 Environment Variables 26
2.7.1 How to Set Environment Variables 26
2.7.2 Common Environment Variables 28
2.8 Terminals 29
2.8.1 The 3270 Display Device 29
2.8.2 Escape Sequences 30
2.8.3 Character Translation 31
2.8.4 If Your Application Is Indefinitely Suspended 34
2.8.5 Support for 132 Column Displays 34
2.9 Editors 34
2.9.1 UNIX Text Editors 35
2.9.2 XEDIT Text Editor 36
2.9.3 Full-Screen CMS and the OpenEdition Shell 43
2.10 Printers 44
2.10.1 UNIX Printing 44
2.10.2 VM/ESA OpenEdition Printing 45

2.11 OPENVM Commands 46
2.11.1 Entering Long OPENVM Commands 46
2.11.2 OPENVM RUN Command 46
2.11.3 OPENVM GETBFS Command 48
2.11.4 OPENVM CREATE EXTLINK Command 50
2.11.5 Limitations of OPENVM Commands 52
2.11.6 OPENVM Command Cross Reference 52
2.12 File Transfer and Archive 53
2.12.1 File Transfer Program 54
2.12.2 TAR 57
2.12.3 CPIO 58
2.12.4 PAX 59
2.13 Programming Extensions for OpenEdition 66
2.13.1 REXX Extensions 66
2.13.2 CMS Pipelines Extensions 66
2.14 Tools Making Life Easy 68
2.14.1 BFSLIST 68
2.14.2 DU 69

Chapter 3. OpenEdition Planning Summary 71
3.1.1 Finding Additional Information 71
3.1.2 Creating Byte File System File Spaces 72
3.1.3 Installing the Shell and Utilities Feature 72
3.1.4 General Preparation 72
3.1.5 Assigning POSIX User IDs to VM Users 72
3.1.6 Defining POSIX User Groups 73
3.1.7 Assigning VM Users to POSIX User Groups 73
3.1.8 Selecting Additional Security Features 73

Chapter 4. Byte File System Installation 75
4.1 File Pool Overview 75
4.1.1 File Pool Structure 76
4.1.2 The File Pool Administration Machine 77
4.2 File Server Planning 78
4.3 Default Byte File System File Pool Servers 78
4.3.1 VMSYS File Pool Server 79
4.3.2 VMSYSU File Pool Server 80
4.3.3 VMSYSR File Pool Server 80
4.4 A New File Pool for BFS Data 80
4.4.1 Definitions Used for a File Pool Server 81
4.4.2 Define the Server Machine 83
4.4.3 Define the Administration Machine 85
4.4.4 Generation of a File Pool 86
4.4.5 File Pool Server Profile EXEC 88
4.4.6 Create a Customized BFS 88
4.4.7 Advantages and Disadvantages of Using a Dedicated BFS Server . . 90
4.5 Shell and Utilities Feature Installation 91
4.6 Creating User File Pool Spaces 94
4.7 Adding Space to a File Pool Server 95
4.8 Restoring the Environment after Installation Errors 96
4.8.1 Errors during BFS Server Generation 96
4.8.2 Restoring the BFS after the Execution of BFSROOT 96
4.8.3 Restoring the BFS after Building the Shell File Trees 96
4.8.4 LOADBFS Tables 96
4.9 A BFS Supported by Multiple File Pool Servers 98

4.9.1 External Links . . . 99
4.10 Mount Points . . . 100
4.10.1 User Home Directories Using External Links . . . 101
4.10.2 Temporary User Mount Points . . . 103

Chapter 5. System Administration . . . 107
5.1 Profiles for the Shell . . . 107
5.1.1 System Profile . . . 107
5.1.2 User Profile . . . 109
5.1.3 Profile Variables . . . 109
5.1.4 Shell Prompts . . . 112
5.1.5 Online Help Alias . . . 112
5.2 File System Space Monitoring . . . 112
5.3 File System Backup and Restores . . . 113
5.3.1 File System Backup Considerations . . . 114
5.3.2 Concurrent File System Backup . . . 116
5.3.3 File System Backup Examples . . . 116
5.3.4 Restoring Individual Files . . . 119
5.4 DFSMS and the Byte File System . . . 122
5.4.1 Inodes for CMS Files . . . 123
5.4.2 DFSMS Installation Considerations . . . 123
5.4.3 Managing a BFS with DFSMS/VM . . . 123
5.4.4 DFSMS/VM Reporting . . . 126
5.4.5 DFSMS/VM Recalling Files . . . 127
5.5 Scanning the Byte File System . . . 128
5.6 Running POSIX Applications in a Saved Segment . . . 129
5.6.1 POSIX Shell in a Saved Segment . . . 129
5.7 Data Sharing in a VM Collection . . . 132
5.7.1 File Access in a Distributed Environment . . . 132
5.7.2 File Processing in a Distributed Environment . . . 132
5.8 API Overview . . . 132
5.8.1 LE/370 Environment . . . 133
5.8.2 OpenEdition Callable Services Library Routines . . . 133
5.9 Servicing OpenEdition for VM/ESA . . . 134
5.9.1 LE/370 and VM/ESA Version 2 Release 1.0 . . . 135
5.9.2 Maintenance User IDs Associated with OpenEdition . . . 136

Chapter 6. User Administration . . . 137
6.1 The Keys to the UNIX Gate . . . 137
6.1.1 What Is a Root . . . 137
6.1.2 What Is a Superuser . . . 138
6.1.3 IDs, Real and Effective . . . 139
6.2 Administering UIDs and GIDs . . . 139
6.3 CP Directory Options and Statements . . . 140
6.3.1 First Time Handling of the Directory Source File . . . 141
6.3.2 Adding POSIX Information to the User Directory . . . 145
6.4 CSL Calls Specific to UIDs and GIDs . . . 149
6.4.1 CSLPSCAN, POSIX Option Reporting Tool . . . 151
6.4.2 DIRPSCAN, Directory Reporting Tool . . . 152
6.5 Creating a Home Directory . . . 153
6.6 Logging, Messaging, and Broadcasting . . . 154
6.6.1 Logging Facilities . . . 154
6.6.2 Messages and Broadcasting . . . 157
6.7 Deleting a POSIX User . . . 157

Chapter 7. Security . . . 159
7.1 Some Basic Concepts . . . 159
7.1.1 Security Policies . . . 159
7.1.2 Who Needs Security? . . . 160
7.1.3 How Much Security? . . . 160
7.2 General Security Structure . . . 161
7.3 Reviewing Your System . . . 162
7.4 VM Extended Security Controls . . . 164
7.4.1 RACF Version 1 Release 10 for VM . . . 164
7.4.2 POSIXOPT QUERYDB . . . 164
7.4.3 Exec Family of Functions . . . 165
7.4.4 mailx Considerations . . . 166

Appendix A. Shell Script Programming . . . 167
A.1 Basic Constructs . . . 167
A.1.1 Variables . . . 168
A.1.2 Shell Script Return Values . . . 171
A.1.3 Control Structures . . . 174
A.1.4 Open and Closing Files . . . 176
A.1.5 Reading from Terminals and Files . . . 176
A.1.6 Writing to Terminals . . . 177
A.1.7 Shell Functions . . . 177
A.1.8 Combining Commands . . . 178
A.1.9 Debugging a Shell Script . . . 179
A.1.10 Shell Extensions . . . 180
A.2 Sample Shell Scripts . . . 180
A.2.1 FS . . . 180
A.2.2 TS . . . 181
A.2.3 BFS . . . 181
A.2.4 TESTFILE . . . 182
A.2.5 COUNT . . . 182

Appendix B. Program Source Code . . . 183
B.1 CSLPSCAN CP Directory POSIX Option Reporting Tool . . . 183
B.2 DIRPSCAN, a Directory Reporting Tool . . . 189
B.3 BFSLIST Program Source Code . . . 192
B.4 UNLOADBF EXEC . . . 196
B.5 DU Program Source Code . . . 197

Appendix C. Special Notices . . . 201

Appendix D. Related Publications . . . 203
D.1 International Technical Support Organization Publications . . . 203
D.2 Other IBM Publications . . . 203
D.3 On Your Local Bookstand . . . 204
D.4 Getting the Redbook Sample Programs from the Net . . . 204

How To Get ITSO Redbooks . . . 205
How IBM Employees Can Get ITSO Redbooks . . . 205
How Customers Can Get ITSO Redbooks . . . 206
IBM Redbook Order Form . . . 207

Glossary . . . 209

List of Abbreviations . . . 213

Index 215

Figures

1. BFS Directory Contents . . . 11
2. An Example of Permission Bits . . . 14
3. Examples of List File Commands . . . 19
4. Interpreting Fields Reported by an ls Command . . . 19
5. Listing the Three Different File Time Stamps . . . 20
6. Listing the Three Different File Time Stamps . . . 20
7. Shell Command Processing . . . 21
8. Example of chmod Command to Allow Shell Script to Run . . . 22
9. Example of Prefixing . . . 23
10. Example of Adding the Current Working Directory to the Search Path . . . 24
11. XEDIT and history Commands . . . 25
12. Using the history and r Commands . . . 25
13. Example of Shell Session on VM Screen . . . 30
14. HEXOUT EXEC: Create Hexadecimal Table . . . 32
15. Sample Output Table from HEXOUT EXEC . . . 33
16. Example of Using the ed Editor . . . 36
17. Example of a Standard XEDIT Session . . . 37
18. Example of a Customized XEDIT Session . . . 37
19. XEDIT Option BFSLINE . . . 41
20. Tab Characters in XEDIT (before Pressing Enter) . . . 42
21. Tab Characters in XEDIT (after Pressing Enter) . . . 42
22. Full-Screen CMS and the OpenEdition Shell . . . 43
23. Example of lp Shell Command . . . 45
24. Example of OPENVM Command . . . 46
25. Example of OPENVM RUN . . . 47
26. OPENVM GETBFS Command (Module Example) . . . 48
27. OPENVM GETBFS Command (Text Example) . . . 49
28. OPENVM GETBFS Command (Binary Example) . . . 50
29. OPENVM CREATE EXTLINK Command Example . . . 51
30. FTP Logon and Help Example . . . 54
31. HELP SERVER Example . . . 55
32. tar Utilization Example . . . 58
33. cpio Utilization Example . . . 59
34. TELNET Login to Remote UNIX Host . . . 60
35. Backup Examples Using cpio, tar and pax . . . 60
36. Results of Previous Three Backup Commands in UNIX . . . 61
37. FTP Utilization to Get Backup Files from UNIX . . . 62
38. OPENVM PUTBFS Example . . . 63
39. Displaying the Contents of Your Backup Files . . . 64
40. Restoring Your Archive Files . . . 65
41. BFSLIST Sample Session . . . 69
42. DU Output . . . 70
43. DU Shell Script . . . 70
44. File Pool Structure . . . 77
45. BFS Server Direct Example . . . 84
46. The VMSERBFS File Pool Server Supporting the VMBFS Name Space . . . 86
47. Server DMSPARMS Example . . . 86
48. FILESERV POOLDEF Example . . . 87
49. Server PROFILE Example . . . 88
50. BFS LOADBFS File . . . 89
51. BFS LOADBFS Modified File . . . 89

52. BFS LOADBFS Standard File . . . 89
53. BFS LOADBFS Modified File . . . 89
54. BFS File Structure Example . . . 90
55. File Contents Used to Add Disk to Server . . . 95
56. Extract from File BFS UNLOADBF Created by UNLOADBF EXEC . . . 98
57. Dual BFS Server . . . 99
58. External Links Providing Shell Access to New File Pool . . . 100
59. Standard Mount Point . . . 101
60. OPENVM MOUNT Command Example . . . 102
61. User Mount Point Example . . . 102
62. Shell ls Command Example Showing External Links . . . 102
63. OPENVM LIST Command Example . . . 103
64. OPENVM MOUNT and Mount Point Flexibility . . . 103
65. OPENVM MOUNT and Mount Point Flexibility (Continued) . . . 104
66. Example of /etc/profile . . . 108
67. Example of User .profile . . . 109
68. Example of /etc/profile . . . 112
69. BFSPACE Utility to Monitor BFS File Spaces . . . 113
70. Example of QUERY FILEPOOL CONFLICT Command . . . 115
71. Example of .SHOW LOCK Commands . . . 116
72. File Pool Backup Using FILEPOOL UNLOAD . . . 117
73. File Pool Backup Using FILEPOOL BACKUP . . . 118
74. Example of FILEPOOL LIST BACKUP Command . . . 119
75. Example of Output Created by FILEPOOL LIST BACKUP Command . . . 120
76. Example of FILEPOOL RELOAD FILES Command . . . 121
77. Example of FILEPOOL FILELOAD Command . . . 122
78. Example of BFS Migration . . . 125
79. Example of DFSMS REPORT . . . 126
80. Example of DFSMS RECALL . . . 127
81. Saved Segment Example, Setting Up the Segment . . . 130
82. Saved Segment Example: Running the Shell . . . 131
83. Example of Superuser Mode . . . 138
84. IDENTIFY and id Command Output . . . 140
85. GID and UID Permissions to a Particular File . . . 140
86. DIRPOSIX Result Sample Directory Entry . . . 142
87. DIRPOSIX Sample LOGFILE Output . . . 144
88. POSIXGROUP Directory Example . . . 145
89. POSIXINFO Directory Example . . . 148
90. POSIXGLIST Directory Example . . . 149
91. POSIXOPT Directory Example . . . 149
92. CSLPSCAN EXEC Sample Output . . . 151
93. DIRPSCAN EXEC Sample Output . . . 152
94. ADDUSER Shell Script . . . 153
95. ADDUSER EXEC . . . 154
96. .sh_history Contents Sample . . . 155
97. Creating a Console Output File . . . 156
98. .sh_history Contents Sample . . . 157
99. Finding Permissions for a Particular File . . . 163
100. Typeset Example . . . 174
101. Example of a Debugging Aid . . . 179
102. Multiple File Search . . . 180
103. Tree Search . . . 181
104. BFS Shell Script . . . 181
105. TESTFILE - (Multiple File Search) . . . 182
106. Count Shell Script . . . 182

107. Sample EXEC to Demonstrate BPX CSL Calls 184
108. Source Code of BFSLIST EXEC 192
109. Source Code of BFSLIST XEDIT 193
110. Source Code of BFSEXAM XEDIT 194
111. Source Code of BFSDEL XEDIT 195
112. (Part 1 of 2) UNLOADBF EXEC 196
113. (Part 2 of 2) UNLOADBF EXEC 197
114. DU EXEC Source 198
115. DU REXX Source 199

Tables

1. Common POSIX Terminology 7
2. Octal Permission Bits Explained 12
3. Default Permissions 17
4. Common Environment Variables Used in the Shell 28
5. Example of Escape Sequences 31
6. POSIX Characters Commonly Translated 41
7. CMS and Shell Command Equivalents 52
8. Estimates of Control Minidisk Sizes 82
9. Equivalent Commands for the LOADBFS Tables 97
10. File Server Backup and Restore Methods 114
11. VMSES/E Component Description 135
12. User IDs Used for OpenEdition Service 136
13. POSIX Groups Added by SYSENTRIES 143
14. POSIX Users Added by SYSENTRIES 143

Preface

OpenEdition for VM/ESA is the VM/ESA implementation of IEEE POSIX standards for system interfaces and threads. Included in these services is a POSIX-compliant file system known as the Byte File System (BFS). BFS, a companion to the CMS Shared File System (SFS), provides a byte-stream view of files as opposed to the existing record-oriented view. The BFS allows data to be organized and used in a UNIX style and format.

OpenEdition Shell and Utilities provides the POSIX.2 command interpreter and command set. The shell is a command processor that is used to run commands, utilities, request services from the system, execute shell scripts, and run POSIX C applications interactively in the foreground or background.

This publication covers a full range of useful topics for those interested in implementing, administering, or simply learning more about VM/ESA′s POSIX functions. A collection of hints and tips, a selection of sample programs, and a wide range of useful knowledge help make this redbook a unique source of information.

If you are marketing, installing, operating, or using the diverse OpenEdition for VM/ESA function, this book is of value to you.

Basic knowledge of VM and the UNIX world is needed.

How This Redbook Is Organized

This redbook contains 232 pages. It consists of a collection of chapters and appendixes arranged in the following manner:

- **Chapter 1, “VM/ESA′s Launch into the UNIX World”**

  In this chapter VM/ESA′s POSIX function is positioned with respect to the open system computing world.

- **Chapter 2, “OpenEdition Primer”**

  In this chapter you will find a discussion of the origins of UNIX and an introduction to many OpenEdition for VM/ESA topics for end users and system administrators.

- **Chapter 3, “OpenEdition Planning Summary”**

  Planning is always part of a successful installation. In this chapter, the primary implementation topics are outlined.

- **Chapter 4, “Byte File System Installation”**

  Installing the Byte File System and Shell and Utilities is well-documented in other publications. However, you may want to read this chapter if you intend to customize your installation. Various advantages and disadvantages regarding a customized installation are discussed.

- **Chapter 5, “System Administration”**

  Administration of the OpenEdition for VM/ESA function is discussed in this chapter. System and user profiles, file system backups, DFSMS file system management, and how to load POSIX code in a shared segment are some of the topics.

- **Chapter 6, “User Administration”**

  This chapter discusses issues that are global to a system, such as the tasks involved in updating the CP directory to give users UNIX attributes. Several useful tools are introduced that will assist you in user management.

- **Chapter 7, “Security”**

  In this chapter, the various topics of security are introduced. You will learn how to review the POSIX file system to look for security exposures.

- **Appendix A, “Shell Script Programming”**

  System programmers and end users are more productive when they know the local language. This language in the UNIX world is called the shell scripting language. While the shell scripting language is similar to REXX, it has various features that make it powerful and unique. Those features are discussed in this section.

- **Appendix B, “Program Source Code”**

  This section contains the source code for the large programs discussed throughout this publication.

- **Appendix C and D**

  The “Special Notices” and, “Related Publications” sections are located in these appendixes.

The Team That Wrote This Redbook

This redbook was produced by a team of specialists from around the world working at the International Technical Support Organization Poughkeepsie Center.

The authors of this document are:

Gustavo Mindreau	IBM Belgium
Rod Nash	IBM Australia
Miguel Otero	IBM Venezuela

This residency was coordinated by:

Scott Vetter	IBM ITSO, Poughkeepsie

We would also like to acknowledge the professionals who took time to review this document and provided invaluable advice and guidance during its development. Without their excellence and commitment, this project would have been impossible:

Mark Antes	IBM Endicott
Kris Buelens	IBM Belgium
Jack Crast	IBM Endicott
Mike Donovan	IBM Endicott
Joel Farrell	IBM Endicott
Sue Farrell	IBM Endicott
Dave Minch	IBM Tucson
Reed Mullen	IBM Endicott
Steve Record	IBM Endicott
Brian Wade	IBM Endicott

Comments Welcome

We want our redbooks to be as helpful as possible. Should you have any comments about this or other redbooks, send us a note at the following address:

redbook@vnet.ibm.com

Your comments are important to us!

Chapter 1. VM/ESA's Launch into the UNIX World

The release of VM/ESA Version 2 Release 1.0 with OpenEdition for VM/ESA in October 1995 was a major step in the evolution of the VM product. VM/ESA, IBM's premier S/390 interactive operating system, now supports the IEEE Portable Operating System Interface for Computer Environments (POSIX) and the Open Software Foundation's Distributed Computing Environment.

OpenEdition for VM/ESA combines the proven capabilities of preceding VM solutions with the additional interoperability and portability achieved through DCE and through selected POSIX standards to become a true enterprise server. OpenEdition for VM/ESA complies with key industry standards that help improve interoperability while increasing portability of tools, applications, data, and people skills. These standards include:

- POSIX 1003.1 System Interfaces
  - POSIX 1003.1a Extensions
  - POSIX 1003.1c Threads
  - POSIX 1003.1d Spawn() Subset
- POSIX 1003.2 Shell and Utilities
- OSF DCE (1.02 level)
  - Remote Procedure Call (RPC)
  - Directory Services Client
  - Security Services Client
  - Threads (through POSIX)

1.1 Application Portability

Application portability is a major concern of many companies. Business dictates that an application that was originally developed for small LAN environments must be quickly and easily portable to an enterprise-wide, enterprise server environment.

The ease of porting to the OpenEdition for VM/ESA platform is illustrated by the fact that many of the most powerful UNIX tools including `awk`, `grep`, `lexx`, `sed`, `pax`, and `tar` were ported with no change from the UNIX-based Mortice Kern Systems, Inc. (MKS) *Interopen/POSIX Shell and Utilities* product during the development of the OpenEdition Shell and Utilities feature.

Application portability does not just refer to the portability of application code, but to the portability of people's skills as well. An enterprise has a major investment in the skill and knowledge of its people. It is very likely that new computer programmers will have a strong UNIX programming background rather than an S/390 background. By supporting POSIX 1003.2 Shell and Utilities, VM/ESA Version 2 Release 1.0 allows your new programmers to have an interface with the look and feel that they recognize. There is no longer a need to invest time and money to retrain those new programmers. With the OpenEdition Shell environment, they can be productive immediately.

Your existing investment in people's skills is also protected because the POSIX shell is viewed by CMS as an application that provides its own subcommand environment. The strong links between CMS and POSIX allow a user to move between environments easily. Because VM commands and user applications are available from the shell environment, you have the combined function of *VM* and *POSIX* utilities and applications in a single interface.

1.2 Interoperability

OpenEdition for VM/ESA's support for DCE gives you the capability to develop and deploy client/server applications that access resources such as files, data, applications, and services that reside on diverse hardware and software platforms. These applications can operate across complex heterogeneous hardware systems and on various software platforms. DCE masks the complexity of the networks and different system architectures and allows the sharing of enterprise-wide resources. DCE allows you to write and launch client/server applications and perform administrative tasks from a centralized point.

1.3 Scalability

As a business continues to grow and evolve, so must a computer system be able to grow and evolve with the business. The ability to grow with a business is called *scalability*. A system that starts life supporting only a handful of users may be required to support tens of thousands of users in the future.

Scalability is the ability to configure a system over a wide range of processing power. This is a particularly important attribute from viewpoint of the enterprise server. As a customer's business needs grow and change, the system must be flexible enough to match those needs.

Many UNIX systems in the marketplace today still suffer from the problem of scalability; that is, they cannot grow to match the changing needs of today's information technology environment.

VM/ESA is one of the most scalable operating systems in the marketplace. VM/ESA provides exceptional performance from the smallest IBM S/390 processor available today, the PC Server 500 S/390, to the largest IBM bipolar processor, the ES/9000 model 9021-9X2. This is more than a hundredfold increase in processing capacity, with the same operating system providing the same level of function across the entire range.

1.4 VM/ESA: The System of Choice

With OpenEdition for VM/ESA, existing VM customers have a choice. In the past, if a user department was looking at a UNIX application, the only option was to install it on another hardware and software platform. Suddenly there was another system that somehow had to be integrated into the existing environment; this brought about many changes. Problems that had to be solved included:

- Programmer training for installation and maintenance
- System operator training

- Help desk operator training
- System management, backup, and recovery
- Security
- Network protocols and integration
- Tape library compatibility

Today the solution to these problems is simple: port the application to OpenEdition for VM/ESA. OpenEdition for VM/ESA allows the seamless integration of POSIX-compliant applications into the traditional VM/ESA structure.

An example of this seamless integration is the ability of DFSMS/VM to manage the new Byte File System (BFS) without requiring any change to the DFSMS/VM product. DFSMS/VM provides the same function with the BFS as it does with the CMS record file system: the ability to have the system manage your storage environment for you, compress inactive data, migrate data to other storage, and automatically recall and decompress data when referenced.

VM/ESA Version 2 Release 1.0 is the system to meet the most diverse VM customer. It provides:

- Enhancements to all traditional features and functions
- Integration of the LANRES/VM and LFS VM products for workstation affinity
- Interoperability with OSF DCE
- Application portability through IEEE POSIX compliance

Chapter 2. OpenEdition Primer

This chapter is for readers who have little UNIX experience or wish to learn more about OpenEdition for VM/ESA function. By combining many CMS commands and tools with the new UNIX-like VM/ESA POSIX implementation, the goal is to make this primer as easy to understand as possible.

Other chapters in this publication are dedicated to operations that are typical of a system programmer or administrator, but they may be of interest to an advanced end user wishing to gain an in-depth understanding of OpenEdition for VM/ESA.

2.1 A Short UNIX History Lesson

In 1965, Bell Telephone Laboratories, the research arm of AT&T, joined General Electric and Massachusetts Institute of Technology′s Project Mac to develop an operating system called Multics. The Multics design goal was the same as VM′s, which was the enabling of many users to access the system simultaneously, and the sharing of data and resources as required. The Multics system took longer to develop than originally planned and in 1969 Bell Labs withdrew from the project.

One of the Bell programmers involved, Dennis Ritchie, was frustrated with the development of Multics. On his free time, he teamed up with a language expert, Brian Kernighan, they located a computer, and within a short time developed the first version of a new operating system they called UNIX. By the early 1970′s, many departments within Bell Labs were introduced to then depended on applications running on this version of UNIX. UNIX spread quickly throughout Bell Labs. Not only did the labs use UNIX, but Ritchie and Kernighan presented a paper on UNIX to the University of California at Berkeley, and it was just what the university was looking for.

Bell Labs was not a computer hardware manufacturer, and they had no control over hardware design. They did not know what features and functions would be brought forward from an old hardware design to a new one, or what new feature or functions would be available on the new machines.

UNIX was originally written in an assembler programming language. Assembler is a language that is very closely linked to the hardware on which it is running. Rewriting an operating system written in assembler to run on even a slightly different hardware platform is a major task. In 1973, most of UNIX was rewritten in the C programming language, to give it portability. Eventually UNIX was running on widely different hardware platforms.

UNIX was originally distributed to colleges and universities as a set of source code and development tools (for a small charge). It was at this time that UNIX started to spread out in all directions, as different groups of people developed functions and features for their copy of the UNIX system.

From this development, two major versions of UNIX have evolved: AT&T Bell Labs System V, and Berkeley Software Distribution (BSD) Version 4. The majority of UNIX operating systems in the marketplace today are based on either System V or BSD V4.

It became obvious during the 1980′s that UNIX applications were not as portable as they were assumed to be. Applications tended to rely on some particular feature or function stemming from a local enhancement to UNIX. There was a need to bring the ever-spreading UNIX operating system functions back together by defining standards which would be universally implemented.

X/Open and the Institute of Electrical and Electronics Engineers (IEEE) are two organizations that are defining standards for interoperability and portability. X/Open is responsible for the UNIX brand, and IEEE developed the POSIX standards.

2.2 OpenEdition Overview

OpenEdition for VM/ESA is part of VM/ESA Version 2 Release 1.0. It extends VM by providing the following integrated functions:

- A POSIX compliant BFS that is a companion to the CMS SFS
- File pool commands to backup BFS data
- OPENVM commands to run programs, migrate data, and manage the BFS
- CP support of the POSIX user database, including tools for administrators
- XEDIT support of the BFS
- CSL routines to access POSIX functions
- REXX and Pipelines extensions to access POSIX functions
- POSIX compliant threads for application multitasking
- Sockets, an industry-accepted protocol for communication
- C language application support provided by a LE/370 RTL

The OpenEdition Shell and Utilities Feature for VM/ESA extends this environment by adding a POSIX.2 compliant shell. The shell extends the OpenEdition VM/ESA function by adding:

- Shell built-in commands
- Shell environment variables
- Shell script (execution) language
- Shell aliases
- Improved access to the Byte File System
- Improved control of the multitasking environment
- Support of set-ID applications
- Support of ported UNIX applications
- Application development utilities

In order to successfully migrate applications to the OpenEdition for VM/ESA environment, they must be recompiled. Included in the Shell and Utilities Feature for VM/ESA is the c89 utility. You must install the IBM C for VM/ESA compiler (5654-033) to use c89

2.3 Introduction to the Byte File System

OpenEdition for VM/ESA has a POSIX-compliant file system called Byte File System (BFS).

The OpenEdition Byte File System consists of an ordered sequence of bytes rather than records. BFS allows data to be organized and used in a UNIX style and format. Like SFS files, BFS files are stored in CMS file pools. Multiple Byte File Systems can be enrolled in the same file pool, and a Byte File System can reside in the same file pool as SFS file spaces.

The POSIX interfaces for BFS directories and files are provided as C library routines in the Language Environment library. For other programs, POSIX functions are provided as a series of Callable Services Library (CSL) routines. The POSIX implementation in OpenEdition for VM/ESA provides Pipelines stage commands and a set of OPENVM commands to support BFS. Also, CMS supports a limited use of BFS files and directories, using the CMS record file system interface.

BFS was derived conceptually from the file systems that are present on UNIX operating systems. Compared to the traditional CMS record-oriented minidisk file system, BFS exhibits different semantics, naming conventions, and file structures.

2.3.1 Terminology

The following is a partial list of some of the file-related terms defined as part of POSIX 1003.1. Some of the definitions are followed by additional VM specific information, interpretation, and restrictions (where applicable).

Table 1 (Page 1 of 4). Common POSIX Terminology

File System				
POSIX	A collection of files and their attributes. Each file system has a logical starting point, called the *root* of the file system, which is denoted with the slash character (/).		In the VM implementation of POSIX, the concept of *fully qualified root* is introduced and defined as: `/../VMBFS:`*`filepoolid`*`:`*`filespaceid`*`/` An OpenEdition for VM/ESA Byte File System is uniquely identified by its fully qualified root. The prefix of `/../VMBFS` is always the same.	**VM**
Historically, the term *file system* has had two meanings: a complete file hierarchy, and a mountable subset of that hierarchy (that is, a mounted file system). POSIX uses the term file system in the second sense.				
In an SFS repository, there can be any number of distinct Byte File Systems.				
It is possible for a process to create (using the OPENVM MOUNT command, the BPX1MNT system call, or external links) larger logical file systems which consist of any number of individual Byte File Systems.				
File				
POSIX	An object that can be written to, or read from, or both. There are different types of files: regular files, character special files, FIFO special files (pipes), sockets, symbolic links, and directories.		In the VM implementation of POSIX, we introduce the concept of *external links*. External links provide a bridge between the OpenEdition for VM/ESA environment and traditional CMS minidisk files and programs. External links also provide implicit mounting of individual Byte File Systems.	**VM**

Table 1 (Page 2 of 4). Common POSIX Terminology

File Name				
POSIX	A name consisting of 1 to NAME_MAX characters used to name a file. The slash character and the null character are not allowed. NAME_MAX is defined in <limits.h> and cannot be less than 14 characters long. The character set for file names depends on the international environment as set by the application, that is, the setlocale() function. POSIX-compliant applications should form names only from the POSIX portable character set. The character set consists of the capital letters A to Z, the small letters a to z, the digits 0 to 9 and the special characters dot (.), underscore (_), and hyphen (-). A compliant name cannot start with a hyphen.		OpenEdition for VM defines NAME_MAX to be 255 characters.	**VM**
Path Name				
POSIX	A string that is used to identify a file. It consists of 1 to PATH_MAX characters, including the terminating null character. Path names have an optional beginning slash, followed by zero or more file names separated by slashes. PATH_MAX is defined in <limits.h> and cannot be less than 255 characters long.		• OpenEdition for VM defines PATH_MAX to be 1023 characters. • If a path name starts with two slashes, but not three, the name is assumed to be a C/370 allowed name for a CMS record file. Control will be passed to the existing ANSI-C routines in the C run-time library. • If a path name starts with a slash-dot-dot-slash (/../), the next component is assumed to be of the type VMBFS:*filepoolid*:*filespaceid* (that is, the path name starts with a fully qualified root). This mechanism provides a means to address files in any one of the Byte File Systems that are stored in SFS servers, without having to change the working directory and root setting of the process. In other words, in the VM implementation of POSIX, it is possible for a process to operate simultaneously on files from disjoint file systems.	**VM**
Path name /A/B/C will be resolved to file C in the process' current file system.				
Path name /../VMBFS:SERVER8:EDL5/A/B/C will be resolved to file C in the POSIX file system EDL5, which is in SERVER8.				
Path name //A B C (or //A.B.C) will be resolved to the CMS record file with file name A, file type B, and file mode C.				

Table 1 (Page 3 of 4). Common POSIX Terminology

Directory				
POSIX	A file that contains directory entries. No two directory entries in the same directory can have the same name, that is, directory entries are unique within a directory. Several directory entries can associate different names with the same file.		VM/ESA supports external links which automatically mount another file system when traversed.	**VM**
Regular File				
POSIX	A file that is a randomly accessible sequence of bytes, with no further structure imposed by the system.		VM/ESA supports the structure of a regular file through the services of the Byte File System.	**VM**
Character Special File				
POSIX	A file that refers to a device. One example of a character special file is a terminal device.		In OpenEdition for VM/ESA, an entry in the file system for a terminal character special file is not required. The system provides a default terminal device which is identified with the name `/../VMBFS:VMOPEN:DEV/TTY` as a special provision. The terminal character special file `/dev/tty` is automatically created during installation.	**VM**
Block Special File				
POSIX	A file that refers to a device. Examples of block special files include tape drives and DASD.		OpenEdition for VM/ESA does not support block special files.	**VM**
Working Directory				
POSIX	A directory, associated with a process, that is used in path name resolution for path names that do not begin with a slash. A process always has a working directory associated with it.		VM/ESA fully supports this concept.	**VM**
User ID				
POSIX	A non-negative integer that is used to identify a system user. When the identity of a user is associated with a process, a user ID value is called a real user ID, an effective user ID, or an optional saved set-user-ID.		VM/ESA defines the real user ID in the CP directory with a new POSIXINFO statement.	**VM**

Table 1 (Page 4 of 4). Common POSIX Terminology				
Group				
POSIX	A non-negative integer that is used to identify a group of system users. Each system user is a member of at least one group. When the identity of a group is associated with a process, a group ID value is called a real group ID, an effective group ID, one of the optional supplementary group-IDs, or an optional saved set-group-ID.		VM/ESA defines the real group ID in the CP directory with a new POSIXINFO statement.	**VM**
Effective User ID and Effective Group ID				
POSIX	The attribute of a process that is used in determining access permissions.		Within the shell, this term refers to the creation of an effective user ID. Superuser is an example of this.	**VM**

For a more complete list of the attributes of the VM/ESA POSIX implementation, refer to *IBM OpenEdition for VM/ESA POSIX Conformance Document*, SC24-5725.

2.3.2 Files and Directories

Knowledge of the file hierarchy is necessary for accessing a file. The complete file name includes a *path* that describes the exact location of the file within the file tree. In a POSIX file system, a convention is used to define the initial branches under which additional data is stored. This organization helps the administration of the file system and the handling of the many files and directories.

One important facility that the file system allows is a symbolic link. A *symbolic link* is a file that points to another file or directory.

Following is a list of the root directories that you are likely to see in a POSIX file system, as implemented in OpenEdition for VM/ESA.

Directory	Contents
bin	Executable commands and their binaries
dev	Files associated with hardware devices
etc	Machine-specific administration files (also known as the administrator's toolbox)
home	User home directories
lib	Points to `usr/lib` libraries and shared libraries
opt	Used for DCE administration files
tmp	Repository for temporary files
u	Points to `/home` (This is an example of a symbolic link.)
usr	Contains many directories such as those for administration, executable programs, libraries, system executable files, and spool (mail)
var	Contains directories for the variable data, such as DCE, logs, spool, and security

Figure 1 shows an abstract representation of a typical BFS file tree.

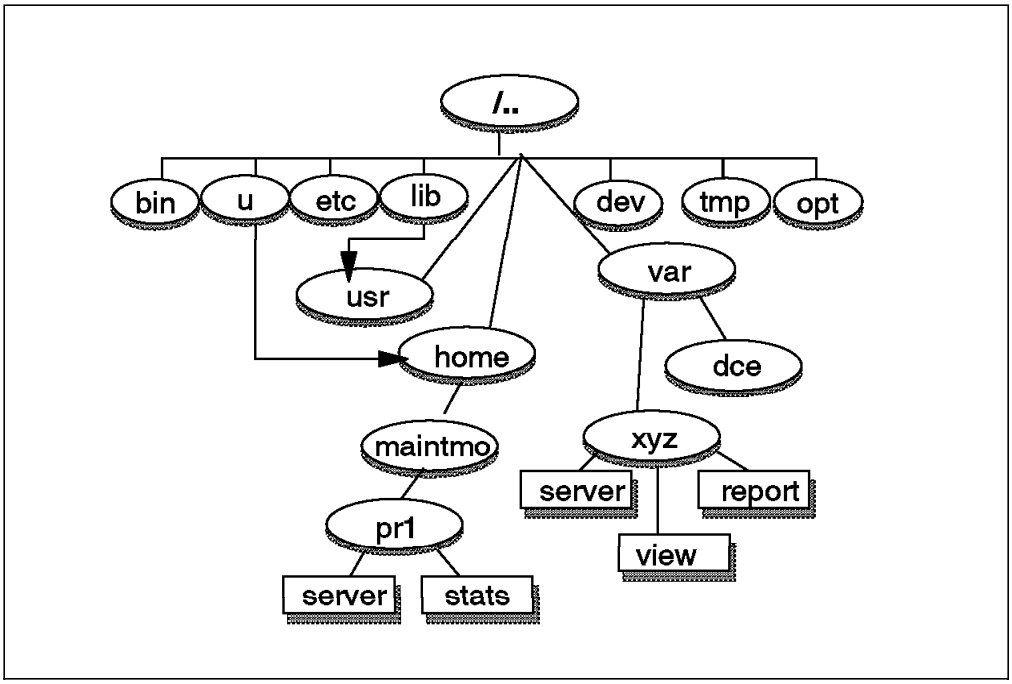


Figure 1. BFS Directory Contents

What kind of name should you give a file? Personal preference plays a key role in naming files. In general, keep them as simple as you can without making them difficult to remember or to type. After all, you want to be able to remember what is in the file when you see its name.

The dash (-), dot (.), and underscore (_) characters, when used within a name, make the name easier to read and may help in sorting and searching.

You cannot use the right slash character (/) in a file name. The following characters have special meanings to the shell interpreter, so do not use them in file names:

\ < > | & ? $ ( ) { } * # / (blank space) ″ ″

You should also avoid starting a file name with a dash (-). And finally, it is acceptable to use a dot (.) in a file name, but if the name starts with a dot, the file name will be hidden; it will not appear in a default list of files.

2.4 Understanding File System Permissions

This section discusses file system permissions and how they are handled in OpenEdition for VM/ESA.

The following topics are covered:

- Permission bits
- Directory permission bits
- Advanced information on permission bits
- Default permissions

In VM there are three unique repositories for user data, namely:

CMS Minidisks You can authorize users to link in either read or write the object to which the granted authority applies. Users have permission to access and execute all of your files with just read permission. Write access implies read access.

SFS files You can authorize users to either *read* or *write* at the file level they are granted access to. The granularity is finer here, but write access still implies read access and users can execute anything they can read.

BFS files Here things are different. You authorize your files *separately* for *read*, *write*, and *execute*. Write does not imply read, and read does not imply execute.

2.4.1 Permission Bits

Files (and directories) have permission bits. You must thoroughly understand these. These are sometimes referred to as *mode* bits, but the term *permission bits* or *permissions* is preferred. The basic permission bits are quite simple.

There are 12 permission bits:

- Three *system* bits (which are not directly displayed)
- Three *owner* bits that describe what the owner is permitted to do
- Three *group* bits that describe what any other user who is a member of the group may do
- Three *other* or *world* bits that describe what any other user can do

Permission bits are often displayed as nine bits. (The three high-order system bits are displayed in special ways). A typical permissions display is:

```
rwxr-xr--
```

The first three characters displayed are the owner bits, the next three are the group bits, and the last three are the other bits. Within each three-bit group, the first bit is for *read* permission, the second for *write* permission, and the last for *execute* or *search* permission.

By convention, a letter means the bit is on, and a dash means it is off. In the example above, the owner has read/write/execute permission, anyone in the file's group has read/execute permission, and everyone else has read permission. Permissions are not hierarchical; write does not include read, and execute also does not include read.

Permission bits are often written in octal. The above example, in octal, is 754. Permission numbers are sometimes called *absolute permissions*. In octal, the first digit represents the owner's permissions, the second digit is the group's permissions, and the third digit is everyone else's permissions. When using octal notation, sometimes four digits are shown. Here, the first digit contains the system permissions (which are explained later). Table 2 shows the different possible permission numbers.

Table 2 (Page 1 of 2). Octal Permission Bits Explained

Octal number	Permission meaning
0	None
1	Execute only
2	Write only

Table 2 (Page 2 of 2). Octal Permission Bits Explained	
Octal number	**Permission meaning**
3	Write and execute
4	Read only
5	Read and execute
6	Read and write
7	Read, write, and execute

The execute permission is not as simple as it might appear:

- For binary programs (produced by a compiler, and linked for execution) it functions in the obvious way.
- Shell scripts are only partly limited by execute permissions. If the name of a shell script is entered on the command line in the normal manner, the execute permissions are examined. If the shell script is named after a *dot* command, or named when starting a new shell, the execute permissions are not examined. In these cases, the read permissions are examined instead. The logic is that the current shell is *reading* the shell script as a data file. It is interpreted, not executed.
- The meaning of execute permission for directories is described in the following section.

Permissions (for files or directories) are not cumulative for the owner, group, and world fields. In general, the owner field provides more permissions than the group field, and the group field more permissions than the world field, but this is not required. For example:

```
-r--rw-rwx 1   gus    xyz   3210   May 3 03:15 mystuff
```

has rather unusual (but valid) permissions. The owner (*gus*) cannot write or execute this file. (He can change the permissions, of course, and add more permissions for himself, but, as shown here, he cannot write or execute the file). Any member of group *xyz* can read or write the file. Anyone else (other than the owner and any members of the group) can read, write, or execute the file.

Stated a different way, the permissions assigned to the owner, group, or world are also restrictions. The owner, for example, is not considered part of the *world*. This aspect can be used to exclude certain users from accessing a file. This can be done by creating a new group containing all the users to be excluded. The file's group-owner is then changed to this new group name. The group permissions are set to '---'. The result is that no member of the group can access the file, even if the file has full access for the rest of the world.

File permission bits are verified when a file is opened. The commands `mv` and `rm` do not open a file (unless the `mv` is to another file system). Thus it is possible to remove a file that you do not have permission to open, when you have write permission for the directory containing the file.

The owner of a file or directory can always change the permission bits. The owner is usually the user who created the file or directory, but ownership can be transferred (by a superuser) to another user. Three permission bits apply to the owner; the owner can set these to protect himself against his own mistakes. For example, the owner of a file can set the owner permission bits to `r-x`. This will prevent the owner from writing in the file. However, the owner can always

change the permission bits for the file (using the chmod command) if he really wants to write in it.

Remember that a file (or directory) owner can change any of the attributes of the file, but only root can change the *owner* and *group* names.

2.4.2 Directory Permission Bits

In UNIX, a directory is a type of file. As a file, it has permission bits of its own. Permissions to read and write in a directory are independent of permissions to read and write the files named within the directory.

A file's permission bits are inspected when opening, reading, writing, updating, or executing the file. Directory permission is required to find a file before opening it for use.

Note that any security information effective in a specific directory is not propagated to lower directories; each directory obeys only its own permission bits.

Directory permissions are especially important for UNIX system files, such as those in */usr*. You should not allow users to add files to */usr* and its subdirectories unless there is a good reason for doing so. You control this by not allowing *write* permission for *others* in any of these directories. This is the default condition in OpenEdition for VM/ESA. You can list directory permissions with the command: `ls -lD` *dirname*, where *dirname* is the path name of the directory you wish to list. Figure 2 shows permission bits for some directories.

```
ls -lD /
total 0
drwxr-xr-x   1 root     system         0 Sep 20  1995 bin
drwxr-xr-x   1 root     system         0 Sep 14  1995 dev
drwxr-xr-x   1 root     system         0 Apr 11 16:58 etc
drwxr-xr-x   1 root     system         0 Apr  4 15:11 home
drwxr-xr-x   1 root     system         0 Oct 19  1995 opt
drwxr-xr-x   1 bin      bin            0 Sep 14  1995 usr
 mindreau /home/mindreau/ $
```

Figure 2. An Example of Permission Bits

To list the files in a directory, you must have *read* permission for the directory. A user with write permission for a directory can create, rename, or remove a file. Directory execute permission is required to access the directory (when looking for a file, for example). A directory cannot be executed. The `x` permission bit controls the ability to search the directory. To `cd` to a directory, or to use it as part of a path name, you must have search permission for the directory.

To summarize, permission bits used with directories have the following meanings:

- *Read* permission (in owner, group, or world fields) permits a user to read the directory, but not the files within the directory. The `ls` command, among many others, reads directories. It will not list a directory unless the current user has *read* access to the directory. *Read* permission in a directory is required to use wildcards when referencing the directory.

- *Write* permission (in owner, group, or world fields) permits a user to add, delete, or change entries in the directory. Entries may include files or directories. A file can be added, deleted, or renamed. An existing file cannot be read or written (unless the user has appropriate permission to the file), but it can be deleted or renamed since these actions take place in the directory, not the file itself. These actions can be taken regardless of the permissions on the files in the directory. In one way or another, a user can subvert any file, lower-level directory, or files in a lower-level directory if he has *write* permission to the directory.
- *Execute* permission is called *search* permission when applied to a directory. It permits the directory to be used as part of an explicit path name. To access file */u/mydir/file3*, the caller must have *search* access to the *root* directory, the *u* directory, and the *mydir* directory. *Search* permission does not permit listing or reading the entire directory; it permits use of a single entry in the directory. You must, by some external means, know the name of the entry (that is, the path name) of the file you wants.
- The *suid* bit is not used.
- The *sgid* bit is also named the *group inheritance flag* when used with a directory. It controls what GID is assigned to new files created in the directory (including new subdirectories). If this flag is set, the GID assigned to the directory itself is used as the GID for any new files created in the directory. If the flag is not set, the GID of a new file is the current group of the user who created the file. A user can change his current group with the `newgrp` command.
- The *sticky* bit means that, even though the directory is writable by the current user, only the owner of a file in the directory, the owner of the directory, and root, can delete (*unlink*) a file. When used this way, the bit is sometimes called the *link permission flag*. Note that this flag also prevents non-owners from renaming a file (with the `mv` command). This flag is commonly used for shared directories, such as */tmp*, and various spool and mail directories.

2.4.3 Advanced Information on Permission Bits

UNIX uses 12 permission bits. Of these, nine are the basic r/w/x permissions for owner/group/other, and were described previously. The three remaining bits are somewhat more complex. They are:

1. The set UID (or *suid*) bit
2. The set GID (or *sgid*) bit
3. The save-text (or *sticky*) bit

These bits are critical for security controls, and are displayed by modifying the normal “rwxrwxrwx” string used to display the basic permission bits. For display purposes, these bits modify the three `x` bits in the normal display.

The *suid*, *sgid*, and *sticky bit* are displayed by changing the `x` in the owner `rwx` to one of the following characters:

s If in the owner permissions section, the *set-user-ID* bit is on; if in the group permissions section, the *set-group-ID* bit is on. The *execute* bit is also on.

S This means the same thing as `s`, except that the *execute* bit is off.

t The *sticky* bit is on. The *execute* bit is also on (ignored under OpenEdition for VM/ESA).

T This means the same thing as t, except that the *execute* bit is off (ignored under OpenEdition for VM/ESA).

These bits are set with the chmod command, using either symbolic operands or a 4-digit octal operand.

The suid Bit

The *suid* bit means that the program will run under the authority of the UID of the owner of the file. (Executable files are normally executed under the authority of the UID of the user who is logged into the system and asking to have the file executed.) For example:

```
-r-sr-xr-x 1   root   sys   3254   Jun 1 11:30 myprog
```

has the *suid* bit set. If I execute *myprog*, it will execute with *root* authority. Since *root* can bypass almost all security controls, this could be dangerous. By executing any program (with *suid* to *root*), you effectively become *root*.

The *suid* bit can be set using the chmod command. It is automatically removed by the cp (copy) command. There is no direct way for a normal user to create a *suid root* file.

The *suid* function can be used by owners other than *root*. It can be used, for example, to ensure that a file is accessed only by a certain program. For example:

```
-rw------- 1   gus    sys   5432   Jun 2 13:45 mydata
-r-sr-xr-x 1   gus    sys   2345   Jun 1 11:30 myprog
```

permits anyone to execute *myprog*. Only user ID *gus* can access *mydata*. Since anyone can execute *myprog*, and since *myprog* uses suid to execute as *gus*, anyone can access *mydata* only by executing *myprog*. The assumption is that this program contains whatever security controls it needs to manage proper access to *mydata*. You should completely understand this example. It incorporates many of the key elements of permission bits, and represents a practical way to control the use of shared data.

The sgid Bit

The set group (*sgid*) function works just like the *suid* function, using a file's group identity instead of the owner identity.

The *sgid* bit has a special meaning when used with a directory where it determines how group ownership for new files is assigned.

The sticky Bit

The *sticky bit* is used for multiple purposes. In earlier systems it was used to indicate that a program should be retained in memory after execution, to improve system performance. This function is not used in modern UNIX systems. Instead, it is used with directories to further limit who may alter entries in the directory.

Note

The *sticky bit* can be set, but OpenEdition for VM/ESA will take no action based on its setting.

2.4.4 Default Permissions

Table 3 shows the default permissions set by the system. You can override the defaults by setting the system mask using OPENVM SET MASK or the umask shell command.

Table 3. Default Permissions

Task	Using	Default Permission
Create a directory	mkdir shell command	rwx rwx rwx
Create a directory	OPENVM CREATE DIRECTORY	rwx rwx rwx
Create a file	XEDIT command	rw- rw- rw-
Create a file	ed editor	rw- rw- rw-
Create a file	Redirection (>)	rw- rw- rw-
Create a file	cp command	Sets the output file permissions to the input file permissions, except suid.
Create a file	OPENVM GETBFS command (when used for BFS to BFS copy)	If any execute permissions are on the source file: rwx rwx rwx If no execute permissions are on the source file: rw- rw- rw-
Create a file	OPENVM PUTBFS command	If the source file is a BFS file and any of its execute permissions are on, or if the source file is a CMS record file in MODULE format: rwx rwx rwx Otherwise, default permissions are: rw- rw- rw-

2.5 Introduction to the OpenEdition Shell and Utilities

The shell environment can be thought of as the user interface to UNIX. There are many different shells in the UNIX world and each has different features, but all of them affect how commands will be interpreted, and provide tools to define your work environment.

The shell is simply a command interpreter; it translates commands into machine specific functions.

A program or user interacting with the shell environment has access to the underlying VM/ESA system. The shell itself is a CMS application that emulates the UNIX shell environment. From the shell environment, any VM command or application residing in the CMS file system may be executed by prefixing the command with cms.

The OpenEdition Shell environment is modeled after the UNIX System V shell with some of the features found in the Korn Shell. It conforms to the POSIX standard 1003.2 - 1992.

Be careful when you use a UNIX book as reference for OpenEdition for VM/ESA. There are so many different varieties of UNIX that have different sets of commands and features that you can easily purchase a UNIX book that has little resemblance to the environment in which you will be working.

The best reference material you can read for information on the OpenEdition Shell environment is the *IBM OpenEdition for VM/ESA User's Guide*, SC24-5727, and *IBM OpenEdition for VM/ESA Command Reference*, SC24-5728. There is only a small gain in understanding through reading about the C shell or the `vi` editor, or studying the wide range of POSIX.2-compliant UNIX commands found in UNIX publications.

2.5.1 Starting the Shell

If the OpenEdition Shell and Utilities is installed on your system and the BFS is installed using the default file pools, then the following commands will start a shell session:

```
OPENVM MOUNT /../VMBFS:VMSYS:ROOT/ /
OPENVM SHELL
```

Exiting from the shell can be accomplished with the following command:

```
exit
```

The OPENVM SHELL command will automatically assure that SCREERUN LOADLIB is on the GLOBAL LOADLIB list before invoking the shell.

The OPENVM SHELL command will interrogate the file `/etc/openvmdefaults`, if it exists, to determine if it needs to access any minidisks or SFS directories to gain access to the C runtime library. It also looks in this file to determine if any other LOADLIBs should be made global. Since this process uses VMLINK, complex startup operations are possible. See section “OPENVMDEFAULTS File” on page 92 for more information.

2.5.2 ls Shell Command

The `ls` command displays a list of the files in the current directory. Several examples of using the `ls` command are shown in Figure 3 on page 19.

```
ls -al
total 24
-rw-------   1 dceadmin system      2078 Apr 19 11:32 .sh_history
-rw-rw-rw-   1 mindreau system         5 Apr 16 14:19 test.test
-rw-rw-rw-   1 mindreau system       226 Apr  4 15:25 vmgreat.books
 mindreau /home/mindreau/ $
ls -ld
drwxrwxrwx   1 dceadmin system         0 Apr 19 11:32 .
 mindreau /home/mindreau/ $
ls -l /u/mindreau/vmgreat.books
-rw-rw-rw-   1 mindreau system       226 Apr  4 15:25 /u/mindreau/vmgreat.books
 mindreau /home/mindreau/ $
ls -ld /u/mindreau
drwxrwxrwx   1 dceadmin system         0 Apr 19 11:32 /u/mindreau
 mindreau /home/mindreau/ $
```

Figure 3. Examples of List File Commands

The first form, `ls -al`, displays information about all the files in the current directory, including *hidden* files (that is, files whose names begin with a period). The second form, `ls -ld`, displays information about the current directory itself. The third form, `ls -l /u/mindreau/vmgreat.books`, displays information about a particular file. The last form, `ls -ld /u/mindreau`, displays information about a specified directory. The general format of the display is shown in Figure 4.

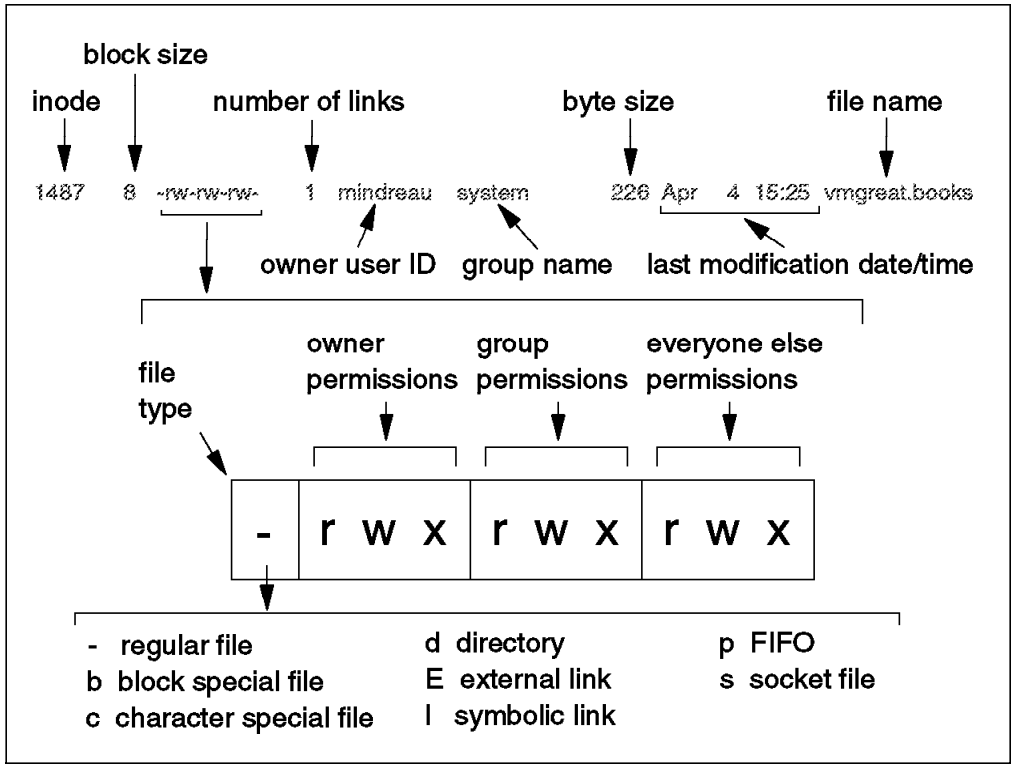


Figure 4. Interpreting Fields Reported by an ls Command

The flag `i` can be used to display inode numbers, but these are not useful in most situations (unless you are a DFSMS user). The complete syntax of `ls` is:

ls [-AabCcDdFfgiLlmnopqRrstuWx1] [*pathname ...*]

2.5.3 File Time Stamps

UNIX systems maintain three time stamps for files (including directories). The time stamps are:

- *atime*. This is the time the file was last accessed. In effect, this is the last time the file was opened.
- *ctime*. This is the last time the *inode* for the file was changed. (It is not the creation time, unless file creation was the last event for the inode.) The *inode* is changed when permissions are changed, the owner is changed, the file size (number of clusters) is changed, and so forth.
- *mtime*. This is the last time the contents of the file were changed. This generally means the file was opened for output.

These times can easily be manipulated by *root* with the `touch` command.

The long forms of the `ls` command normally list the mtime. The `-c` flag can be used to list the *ctime* instead. The `-u` flag can be used to list the *atime*. Figure 5 shows an example of the three different outputs. Also, the `find` command can reference all three time stamps.

```
Monday April 22 01:05:10 PM EDT 1996
 mindreau /home/mindreau/ $
ls -l
total 16
-rw-rw-rw-   1 mindreau system         5 Apr 16 14:19 test.test
-rw-rw-rw-   1 mindreau system       226 Apr  4 15:25 vmgreat.books
 mindreau /home/mindreau/ $

ls -lc
total 16
-rw-rw-rw-   1 mindreau system         5 Apr 16 14:19 test.test
-rw-rw-rw-   1 mindreau system       226 Apr 19 11:32 vmgreat.books
 mindreau /home/mindreau/ $

ls -lu
total 16
-rw-rw-rw-   1 mindreau system         5 Apr 19 11:33 test.test
-rw-rw-rw-   1 mindreau system       226 Apr 16 15:19 vmgreat.books
 mindreau /home/mindreau/ $
```

Figure 5. Listing the Three Different File Time Stamps

If the date of a file is more than 6 months old or if the date is in the future, the year is shown instead of the time, as in Figure 6.

```
ls -l
total 8
-rw-rw-rw-   1 maintsv  system        81 Jan  2  1999 a.a
drwxr-xr-x   1 root     system         0 Sep 20  1995 bin
drwxr-xr-x   1 root     system         0 Sep 14  1995 dev
drwxr-xr-x   1 root     system         0 Apr 29 15:51 etc
drwxr-xr-x   1 root     system         0 Jul 24 15:26 home
```

Figure 6. Listing the Three Different File Time Stamps

2.5.4 Shell Command Processing

When a command is entered to the shell, the shell will search the following places in this order.

1. Built-in commands and aliases
2. Functions
3. Utilities
4. Path search

If an incorrect command is entered, you will receive a “not found” response. If a valid command is entered but contains an incorrect option, sometimes the command will be processed without the options. Figure 7 shows examples of an incorrect option followed by an undefined command.

```
 rodnash // $
ls -x    1
bin   dev   etc   home  krb5  lib   lst   opt   tmp   u     usr   var
 rodnash // $
lx       2
lx: not found
 rodnash // $
```

Figure 7. Shell Command Processing

1 The `ls` command does not recognize `-x`, but executes anyway and gives no warning.

2 The `lx` command is not valid and cannot be found in the current path search order.

2.5.5 Case Sensitivity Within the Shell

In this publication you will see many examples of UNIX commands, such as:

```
ls -AabCcDdFfgiLlmnopqRrstuWx1 pathname ...
```

Type all commands just as they appear. Use the same capitalization as in the examples. UNIX commands respect case. In some instances the upper and lower character will have the same meaning in a command, and other times they will not. The inconsistencies are usually not harmful (you will not suddenly erase all your files), but they can be frustrating. For example:

`ls -l` Displays permissions, links, owner, group, size, time, and name.

`ls -L` Follows symbolic links.

`ls -r` Sorts in reverse of usual order.

`ls -R` Lists subdirectories recursively.

Another area where new users get into trouble is that UNIX file names and directories can also be in mixed case. For example:

```
/home/cartoons/bunny.biped
```

is a distinctly different file from:

```
/home/cArtoONs/BuNNy.biPED
```

2.5.6 Running Programs in the Shell

As you become familiar with the shell environment, you will probably start writing some shell scripts and small C programs. Writing them is easy using XEDIT, but getting them to run may be troublesome.

Execute Permissions

The default file permission bits of a new file do not allow execution. This is easy to fix. Use the chmod command to alter the permission bits to allow execution. Use the umask command to set file permissions at the time of file creation. The umask command is only in effect for the current shell session. The umask command may be added to your .profile file so your file creation permissions are always set on entry into the shell environment.

See section 7.3, “Reviewing Your System” on page 162, for information regarding the proper permission bits to use.

Figure 8 shows the use of the chmod command to update the execution bit to allow a sample shell script to run. The myscript.scp file is run with the default permission bits, which does not allow execute privileges.

```
 maintsv /home/maintsv/ $
ls -l myscript.scp
-rw-rw-rw-   1 maintsv  system        32 Jul 11 09:36 myscript.scp
 maintsv /home/maintsv/ $
./myscript.scp
./myscript.scp: GSU92O9 !cannot execute
 maintsv /home/maintsv/ $
chmod u=rwx myscript.scp
 maintsv /home/maintsv/ $
ls -l myscript.scp
-rwxrw-rw-   1 maintsv  system        32 Jul 11 09:36 myscript.scp
 maintsv /home/maintsv/ $
./myscript.scp
Hello World!
```

Figure 8. Example of chmod Command to Allow Shell Script to Run

Execution of File Not in Current Path

Often, the directory that your script or program resides in is not in your current path search order.

Suppose, for example, a colleague asks for a copy of one of your programs and you copy it into their directory. To make sure the program works properly in its new home, you change your working directory to where you copied the file and you try to execute it. However the program fails because the directory you copied it to is not in your current path search order.

By simply prefixing the program name with ./ the program will execute from your current working directory, whether it is in your current path search order or not. Figure 9 on page 23 is an example of using the ./ prefix:

```
 rodnash /home/mindreau/ $                                          1

ls -l rr                                                            2
-rwxrwxrwx   1 rodnash  system      3578 Apr 24 10:08 rr
 rodnash /home/mindreau/ $

rr                                                                  3
rr: not found
 rodnash /home/mindreau/ $

env                                                                 4
MAIL=/usr/mail/rodnash
PATH=/bin:/usr/bin:/etc:/u/rodnash:/u/rodnash/bin
OLDP=$
SHELL=/bin/sh
_=/bin/env
MAILMSG=You have new mail.
LOGNAME=rodnash
HOME=/u/rodnash
_EDC_KEEP_EMSG=Y
TZ=EST5EDT
 rodnash /home/mindreau/ $

./rr                                                                5
        ┌──────────────────────────────────────────────┐
        │                                              │
        │             Hello - Redbook Readers          │
        │                                              │
        └──────────────────────────────────────────────┘
 rodnash /home/mindreau/ $
```

Figure 9. Example of Prefixing

1 The shell prompt displays the current working directory as /home/mindreau/ and the user ID as rodnash.

2 The list of the current working directory shows that program rr exists.

3 An attempt to run a program that cannot be found is shown here.

4 Use the env command and check the current search path. Directory /home/mindreau is not in the current path search order.

5 By using ./ before the program name, the shell will execute the program from the current working directory.

Adding a New Path to Be Searched

Another way around the search path problem is to add your current working directory to your search path. This is done with the PATH=::$PATH or PATH=:.:$PATH shell command. This command will allow you to execute any program from the current working directory.

If you run the shell with your current working directory in your path, beware that you may execute or access programs designed to take advantage of insecure systems.

Figure 10 is an example of adding your current working directory to your search path:

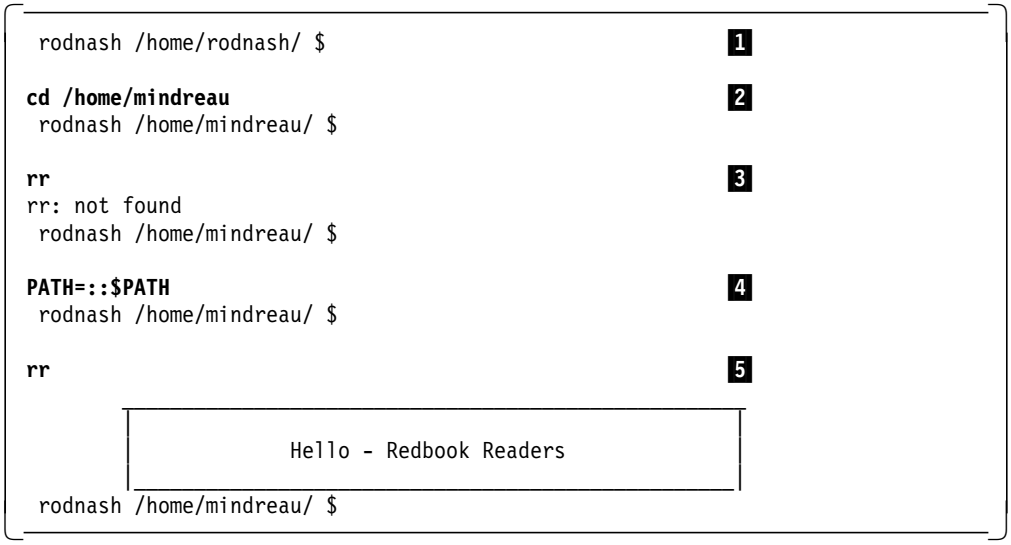


Figure 10. Example of Adding the Current Working Directory to the Search Path

1 The shell prompt displays the current working directory as /home/rodnash and the user ID as rodnash.

2 Change current working directory to /home/mindreau.

3 The program is not found because the current working directory is not in the search path.

4 Enter PATH=::$PATH command.

5 The program now runs.

2.5.7 Using the Shell History File

The POSIX shell automatically saves a list of commands entered. This list is stored in a file in the user's home directory. The name of this file is set by the shell variable HISTFILE, and it is .sh_history by default. history keeps the last 128 commands.

The history and r commands are built-in shell aliases for the UNIX command fc.

The history Command

The history command displays, edits, and re-enters commands that have been used as input during a shell session. A nice feature of history is that it allows you to edit a command or a range of commands with certain available editors.

In the OpenEdition Shell environment, the editors available for history are ed or sed. However, the experienced VM user will find ed or sed frustrating to use. You can use XEDIT to manipulate the stored commands from history, but to call XEDIT directly you need to code a shell script and set the editor to that. Section "Invoking XEDIT to Edit a BFS File" on page 38 shows a sample shell script to accomplish this. Figure 11 on page 25 shows how to use XEDIT to manipulate commands from history:

```
 rodnash /home/rodnash/ $

history -n 30 40 > lastcommands                          1
 rodnash /home/rodnash/ $

x lastcommands                                           2
 rodnash /home/rodnash/ $

. lastcommands                                           3
cmd 30
cmd 31
cmd 32
cmd 33
cmd 34
cmd 35
cmd 36
cmd 37
 rodnash /home/rodnash/ $
```

Figure 11. XEDIT and history Commands

1 This history command will place commands 30 through 40 in file lastcommands without the commands being numbered.

2 The shell script x is used to XEDIT the lastcommands file. (The command fc -e x 30 40 could be used to directly enter the XEDIT environment and immediately execute the commands after the edit session is over.)

3 The dot (.) command will execute the lastcommands file.

The r Command

The r command is used to retrieve and execute commands. There is limited editing capability provided by the r command which can be very useful when working with long commands. Figure 12 is an example of using the r command.

```
 rodnash /home/rodnash/ $

find / -name *dump*.* -mtime +10 -follow -type f -print > /tmp/results   1
 rodnash /home/rodnash/ $

r dump=DUMP                                                              2
find / -name *DUMP*.* -mtime +10 -follow -type f -print > /tmp/results   3

 rodnash /home/rodnash/ $
```

Figure 12. Using the history and r Commands

1 A long find command to look for all files with the characters “dump” in their file name.

2 The r command with an option to change dump (lowercase) to DUMP (uppercase) is entered.

3 The resulting altered retrieved find command is shown.

2.6 Help Is Always at Hand

It may take some time before you are comfortable using the shell commands and their structure. Help is a short command away, even while inside the OpenEdition Shell.

Online help is available for all the shell commands, just as it is for the CMS and CP commands. Enter HELP OSHELL from CMS or `cms help oshell` from within the shell to display a menu of help subjects.

Section 5.1, "Profiles for the Shell" on page 107, describes the sample shell profile which all users use. It has an alias that defines a command `help`, which actually executes the CMS HELP OSHELL command. This alias is shown as follows:

```
alias help="cms help oshell"
```

This command allows a shell user to enter `help` and go directly to the shell CMS help menu from inside the shell.

Also useful is an alias that allows you to go directly to the help menu on the OPENVM commands. This alias is shown as follows:

```
alias hopvm="cms help openvm"
```

This command can be implemented in the user `.profile`. Refer to section 5.1.2, "User Profile" on page 109 for information on how to do this.

2.7 Environment Variables

Environment variables act as a way to control the settings of applications. Their use is generally not limited to a single application, but to all tasks running within a single user ID. This section is a discussion of environment variables.

2.7.1 How to Set Environment Variables

There are several ways to set environment variables. This section describes the most common ways:

- From the shell, using the `export` command
- From CMS, using the CENV group in GLOBALV
- From CMS, by specifying the `ENVAR` runtime option when running a POSIX application

For most CMS users, setting environment variables in GLOBALV is the most convenient method. For shell users, setting environment variables with the `export` command is easiest.

Setting Environment Variables from the Shell

If you are using the shell, you can set environment variables using the `export` shell command. This command adds an environment variable setting that lasts for the current shell session, and is inherited by all POSIX programs and subshells run from that session.

Use the export command as follows:

```
export var1=value1 var2=value2
```

For example, to set the environment variable TZ (time zone) to EST5EDT, use the following shell command (note there is no space between the name of the variable or the value to be set and the equal sign that separates them):

```
export TZ=EST5EDT
```

The TZ variable is set until the current shell session ends or until you reset the variable. To verify that the environment variable is set, you can use the echo command with the $ character preceding the target environment variable. For the previous time zone example, the command would be:

```
echo $TZ
```

For more information on time zones, see the section entitled “Time Zones” on page 110.

You can also use the env command to see the value of all the environment variables for the current shell session.

If there are environment variable settings you want to have in effect for every shell session, you can add export commands to your .profile in the BFS. You can also set environment variables for every shell session by adding them to GLOBALV, which is described in the next section.

Setting Environment Variables from CMS Using GLOBALV

The CMS global variable facility (GLOBALV) can be used to store persistent or temporary environment variable settings. Environment variable settings contained in GLOBALV are automatically made part of each initial POSIX program run by a user. An initial POSIX program is a POSIX program that is run either as a CMS command or by the OPENVM RUN command.

Environment variable settings contained in GLOBALV are also automatically included in the initial environment created for your shell session when you start the shell.

To save an environment variable that will be in effect only for the current IPL of CMS (which is like using the export command from your shell session) issue the GLOBALV command as follows:

```
globalv select cenv setl variable value
```

For example, to set the TZ environment variable to EST5EDT for this IPL of CMS, use the following GLOBALV command:

```
globalv select cenv setl TZ EST5EDT
```

To save an environment variable setting in GLOBALV to last across an IPL of CMS (which is similar to putting the export command in your .profile from the shell), issue the GLOBALV command as follows:

```
globalv select cenv setlp TZ EST5EDT
```

Note that setlp was used in this case, instead of setl. The setlp operand of the GLOBALV command causes the setting to be permanent.

To check the environment variable settings saved in GLOBALV, use the following command:

```
globalv select cenv list
```

One thing you will notice if you enter GLOBALV commands from the CMS command line is that the environment variables settings saved in GLOBALV will always be in upper case. Often you will want to set environment variables with mixed case names or values. You can avoid CMS's normal uppercasing by issuing the GLOBALV command from a REXX EXEC with the REXX command designation set to ADDRESS COMMAND. Commands issued this way are not automatically uppercased. Be sure to correctly code the mixed case portions of the command.

As long as an environment variable setting remains in GLOBALV, it is made part of the environment of all POSIX programs run as CMS commands or run with the OPENVM RUN command. Note that the environment variable settings obtained from GLOBALV may be overridden by the other methods of setting environment variables described in this section.

Setting Environment Variables from CMS Using the ENVAR Option

Another way you can set environment variables when running a POSIX program as a CMS command is to specify the Language Environment ENVAR runtime option. For example:

```
MYPROG POSIX(ON) ENVAR('var1=value1,var2=value2') / pgm-parms
```

Environment variable settings specified with the ENVAR run-time option override those saved in GLOBALV.

The Language Environment ENVAR option is not available if you run your POSIX applications with the OPENVM RUN command, or if you are running from within the shell.

2.7.2 Common Environment Variables

Table 4 gives a list and short description of the shell environment variables.

Table 4 (Page 1 of 2). Common Environment Variables Used in the Shell

Variable	Purpose
-	If a program runs a child shell, this contains the full path name of the calling program.
~	Tilde expands to the value of the home directory.
CDPATH	cd will search the paths specified by CDPATH.
COLUMNS	Sets the output width of several commands.
EDITOR	Commands invoking editors invoke the one named by EDITOR. XEDIT may be called as a default editor.
ENV	sh performs parameter substitution on this value and uses the result as the name of an initialization file or login script.
FCEDIT	This is the name of the default editor used by fc.
HISTFILE	This is the path name used by the history file.
HISTSIZE	The maximum number of commands that the shell keeps in the history file.
HOME	This is the path of your home directory.

Table 4 (Page 2 of 2). Common Environment Variables Used in the Shell	
Variable	**Purpose**
IFS	These are the internal field separator characters.
LANG	This is the default locale used.
LC_ALL	This is the locale used to override any values of LC_ variables.
LINENO	This is the current line number run by the shell.
LINES	Defines the numbers of lines on the output device. Most commands ignore this value.
LOGNAME	Contains your login name, or name of your VM user ID by default.
MAIL	This is the path of the system mailbox.
MBOX	This is the path of your personal mailbox.
MAILCHECK	This is the number of seconds to pass before the system checks for mail.
MAILPATH	This is a list of mailbox files.
OLDPWD	Contains the name of the directory in which you were previously working.
PATH	Contains a list of directories that the system searches to find executable commands.
PID	This is the current process ID of the parent shell.
PS1-PS4	Contains the values of prompt strings used during shell functions.
PWD	This is the name of the working directory.
RANDOM	Returns a random integer. Setting this variable creates a new seed.
SECONDS	This shows the number of seconds elapsed during the current shell session.
SHELL	Contains the full path name of the current shell program.
TMOUT	This shows the number of seconds before the shell ends. This value is reset when the user is active.
TZ	Contains a value used to calculate the current time zone.

2.8 Terminals

The 3270 family of display devices is not keystroke-reactive, that is, the host computer does not react in real time to each keystroke you type. Rather, a 3270 device transmits blocks of data to the host computer only when you press one of a limited set of keys, such as Enter or a PF key. Therefore, when you access the OpenEdition Shell, your terminal will not behave as a normal UNIX terminal. It will not react to keystroke combinations (also known as *escape sequences*). In VM, nothing will happen until you press either the Enter key or one of the function keys.

2.8.1 The 3270 Display Device

Figure 13 on page 30 shows the general layout of a typical OpenEdition Shell session.

```
$
ls -l etc
total 128
-rwxrwxrwx   1 dceadmin system        15 Apr  2 10:33 bfs
-rw-r--r--   1 bin      bin         1199 Sep 14  1995 mailx.rc
-rw-rw-rw-   1 dceadmin system      1893 Apr  3 14:54 profile
-rw-r--r--   1 bin      bin          654 Sep 14  1995 profile.sample
drwxr-xr-x   1 dceadmin system         0 Sep 14  1995 samples
-rw-r--r--   1 bin      bin         4353 Sep 14  1995 startup.mk
-rwxrwxrwx   1 dceadmin system        29 Apr  2 10:30 x
-rw-r--r--   1 bin      bin        11682 Sep 14  1995 yylex.c
-rw-r--r--   1 bin      bin        20943 Sep 14  1995 yyparse.c
$
id
uid=1002(mindreau) gid=1(staff) groups=0(system)
$

                                                          RUNNING   TOTVM1
```

Figure 13. Example of Shell Session on VM Screen

You pass commands to the shell by typing in the *input area* and pressing the Enter key. Responses are displayed in the *output area* of the screen. The lower right corner of the screen contains the *status area*. It includes two words:

Screen state Status of what is happening in the CMS session.

System ID Network node by which this system is known. This portion of the status area never changes.

The five different screen states are:

Running This means the virtual machine is running.

More... This indicates that the screen should be cleared to display more output.

Holding If you press the Enter key when in the More... status, you will be placed in the Holding screen state. You will also go into the Holding state when you receive a virtual console message from another user ID, and there is additional data left to be displayed.

Not Accepted This means the system, generally the Control Program (CP), is busy and is not able to process your input. This status will remain for only three seconds.

VM READ This means line-mode read was requested from your virtual console. You must supply a line of input and then press Enter. After that, your virtual machine will return to the Running state.

If you need more information on how to operate your 3270 session, see *VM/ESA: Virtual Machine Operation*, SC24-5759.

2.8.2 Escape Sequences

The 3270 family of devices all share an EBCDIC code page. Typical UNIX displays use an ASCII-based character set. Escape sequences are used to emulate all of the special key values expected by a UNIX application, using a 3270 device for input. Escape sequences are used for *control* (or portable) characters.

Control characters are keyboard sequences that are processed in a special way. They are used to erase input, to stop and restart output, and to send signals. You enter an escape sequence by prefixing a control character with a cent sign (¢). Table 5 on page 31 contains some examples of this. In this publication you see the notation <EscChar-C>; it means to type the EscChar (by default ¢), type the C character, and then press Enter.

Note: The default escape character (¢) is fixed and cannot be changed.

Table 5. Example of Escape Sequences

Control Character	ASCII Control Sequence	OpenEdition Escape Sequence
<backspace>	control-H	¢h or ¢H
<carriage-return>	control-M	¢m or ¢M
<NAK>	control-U	¢u or ¢U
<DC3>	control-S	¢s or ¢S
<EOT>	control-D	¢d or ¢D

For more information on the escape sequence, see *IBM OpenEdition for VM/ESA: User's Guide*, SC24-5727.

Also remember that you can use the stty command to set your terminal control characters. Some valid operands of the stty command are:

QUIT char Set QUIT character to *char*

ERASE char Set ERASE character to *char*

KILL char Set KILL character to *char*

For example, you could change the default kill character <EscChar-U> to just % by entering the following command: `stty kill %`

Regarding erase and kill, you may use stty to control the enabling of canonical input. If ICANON is set (which is the default), it turns on canonical processing allowing the erase and kill edit functions.

2.8.3 Character Translation

The OpenEdition Shell uses the EBCDIC Latin 1/Open System Interconnection Code Page 1407. You may need to customize your keyboard to get the correct characters interpreted.

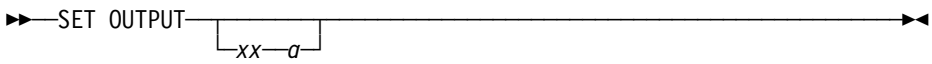
Since C source programs use square brackets for subscripting variables and braces for delimiting program blocks, it can become difficult to understand C source code if these characters do not map out correctly to your country's code page. These special C characters are stored internally with the following hexadecimal representation:

```
[ = X'AD'
] = X'BD'
{ = X'C0'
} = X'D0'
```

With CMS, you can use the SET OUTPUT command to change the representation of hexadecimal values shown in XEDIT, while using the TYPE command to view

the contents of a file, or the cat command when in the shell. The format of the SET OUTPUT command is:

CMS SET OUTPUT Command Syntax



OUTPUT *xx a*

This translates the specified hexadecimal representation *xx* to the specified character “a” for all *xx* characters displayed at the terminal.

OUTPUT

If neither *xx* nor *a* are specified, then OUTPUT returns all characters to their default translation.

For example, if you enter SET OUTPUT C1 a, then your uppercase ′A′ (X′C1′) will now display as lowercase ′a′.

To use SET OUTPUT effectively, first determine which characters are being displayed incorrectly, and then determine those characters′ hex encoding in the 1047 code page. Then, for each character, issue the SET OUTPUT *xx a*, where *xx* is the hex encoding you found and *a* is the way you want the output to appear.

The *HEXOUT* EXEC shown in Figure 14 produces a 12 by 16 table representation of the characters X′40′ to X′FF′. Although the output is translated depending on your code page, it should look similar to the sample in Figure 15 on page 33.

```
00001  /* Create output file 12 by 16 of hex characters from 40 to FF.       */
00002  /*                                                        (SG24-4747) */
00003  Address "CMS"
00004  Write="EXECIO 1 DISKW HEX OUT A (STRING"  /* Init Write command       */
00005  col=4  /* <-- Vertical labels will start at '4' (for x'40')           */
00006  Address "CMS"
00007  Write "   0 1 2 3 4 5 6 7 8 9 A B C D E F"
00008  Write " +---------------------------------+"
00009  Do x=64 To 255                          /* Decimal numbers for loop */
00010     out=d2x(col)"|"                      /* Row delimiter            */
00011     If x=64 Then out=out" "              /* Add blank when x'40'     */
00012     Do y=1 To 16                         /* 16 values per row        */
00013        out=out Strip(x2c(d2x(x)))        /* decimal2hex, hex2char    */
00014        x=x+1                             /* Increase 'x'             */
00015     End
00016     x=x-1                                /* To compensate for loop   */
00017     Write out "|"d2x(col)                /* Write row with labels    */
00018     col=col+1                            /* Increase vertical label  */
00019  End
00020  Write " +---------------------------------+"
00021  Write "   0 1 2 3 4 5 6 7 8 9 A B C D E F"
00022  "FINIS HEX OUT A"                       /* Close the output file    */
00023  "XEDIT HEX OUT A (NOPROF"
```

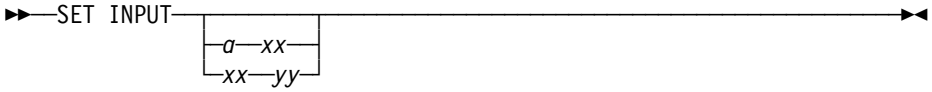
Figure 14. HEXOUT EXEC: Create Hexadecimal Table

```
00001    0 1 2 3 4 5 6 7 8 9 A B C D E F
00002  +--------------------------------+
00003 4|    â ä à á ã å ç ñ ¢ . < ( + | |4
00004 5| & é ê ë è í î ï ì ß ! $ * ) ; ¬ |5
00005 6| - / Â Ä À Á Ã Å Ç Ñ ] , % _ > ? |6
00006 7| ø É Ê Ë È Í Î Ï Ì  : # @ ' = " |7
00007 8| Ø a b c d e f g h i « » ‰ – · · |8
00008 9| ° j k l m n o p q r — Œ æ Ÿ Æ • |9
00009 A| ·  s t u v w x y z ¡ ¿ œ Y ‡ „ |A
00010 B| " £ ¥ fi fl § ¶ ¼ ½ ¾ [ ] ' ' " † |B
00011 C| { A B C D E F G H I     ö ò ó õ |C
00012 D| } J K L M N O P Q R · û ü ù ú ÿ |D
00013 E| \  S T U V W X Y Z · Ô Ö Ò Ó Õ |E
00014 F| 0 1 2 3 4 5 6 7 8 9 · Û Ü Ù Ú " |F
00015  +--------------------------------+
00016    0 1 2 3 4 5 6 7 8 9 A B C D E F
```

Figure 15. Sample Output Table from HEXOUT EXEC

Similarly, for input operations you can use the CMS SET INPUT command to translate characters that you enter to the proper hexadecimal code expected by C.

CMS SET INPUT Command Syntax



Operands

INPUT *a xx*

This translates the specified character *a* to the specified hexadecimal code *xx* for characters entered from the terminal.

INPUT *xx yy*

This allows you to reset the hexadecimal code *xx* to the specified hexadecimal code *yy* in your translation table.

INPUT

This returns all characters to their default translation.

For example, to translate what the US code page sees as square brackets to a POSIX C code page, place these lines in your PROFILE EXEC:

```
"SET INPUT BA AD"
"SET INPUT BB BD"
```

CP Line Mode Considerations

- The default CP LINEDEL character conflicts with the shell default escape character (¢). You must enter TERMINAL LINEDEL OFF before invoking the shell (of course you can also change this LINEDEL to any other character of your preference).
- The default CP ESCAPE character is " (double quotation mark). This can also interfere with some shell commands, such as `awk`.
- The default CP LINEND character is # (number sign). This can interfere with some shell commands, such as `awk`.

For more information on the TERMINAL command, see *VM/ESA: CP Command and Utility Reference*, SC24-5773.

Note: Terminal settings can be overridden individually in the VM directory or globally in SYSTEM CONFIG, using the CHARACTER_DEFaults statement. See *VM/ESA: Planning and Administration*, SC24-5750 for more details.

2.8.4 If Your Application Is Indefinitely Suspended

If your application indefinitely suspends, try one of the following procedures:

- Enter <EscChar-V> or <EscChar-C>.
- Type the HX immediate command (at the first column of the input area) and then press Enter.
- Your last choice is to IPL CMS. Press the PA1 key and then enter IPL (or IPL CMS, if the short version fails).

  Note: Entering #CP IPL CMS will not work if you redefined your terminal LINEND character.

2.8.5 Support for 132 Column Displays

If you have a 25 by 132 display, enter the following shell command or place it in your `.profile` to take advantage of your screen capabilities: `export COLUMNS=132`

2.9 Editors

Text editors are programs that let you create files of text and edit them. They contain just text; there are no colors, formatting controls, font specifications, or graphics.

The most common UNIX text editors are `ed`, `vi`, and `emacs`. In CMS you have one of the most powerful text editors, XEDIT. Even though word processors or desktop publishing programs are available in the UNIX world, you may need to use a simple editor to do things like:

- Create or edit shell script files to add a new function in your UNIX environment.
- Write notes or mail to other users on your system or the world.
- Edit special system files that control your particular UNIX environment.
- Write REXX, Assembler, C, or another type of program.

2.9.1 UNIX Text Editors

Of the hundreds of UNIX commands that exist, the commands for editing files are probably the most used. In the following sections, the most popular UNIX text editors are introduced.

ed Editor

The ed program is the original UNIX editor and is included with the OpenEdition Shell. ed is a *line editor*, which means that the program assigns and manipulates text by line numbers. Every time you want to perform an action, you must specify which lines to do it to. If you are familiar with DOS, you will see similarities to the EDLIN program.

To begin using ed, type a line such as: `ed vmgreat.books`. All ed commands are just one letter long and almost all are in lowercase. ed operates on a *copy* of the file you are editing and hence changes you make to that copy just exist in a temporary file (called a buffer) until you enter `w` to write the changes to disk.

OpenEdition Shell ed Editor: The format of the ed command is:

ed [**-bsx**] [**-p** *prompt*] [*file*]

ed supports the following options:

-b Lets you edit larger files by restricting the amount of paging dedicated to holding records of the file. This paging is located in the `/tmp` directory, by default.

-p *prompt* Displays *prompt* as the prompt string.

-s Puts ed into a *quiet* mode,

- **e, E, r,** and **w,** subcommands do not display file size counts
- the **q** and **e** subcommands do not check buffer modification
- **!** is not displayed after calling the shell to run a subcommand.

-x Runs an **X** subcommand to handle encrypted files properly. This option is currently disabled.

ed accepts the following subcommands:

a Appends text after the specified line.

d Deletes the addressed range of lines.

h Provides a brief explanation of the last error that occurred.

i Inserts text before the specified line.

p Displays the addressed lines.

q Causes the editor to exit.

u Rolls back the effect of the last subcommand that changed the buffer.

w Writes the addressed lines of the buffer to the named file.

Figure 16 on page 36 shows an example ed session. The editor returns the number of bytes found in the file. The **,p,** subcommand displays the contents of the file. The q subcommand ends the session.

```
$
ed vmgreat.books
226
,p,
SC24-5773  VM/ESA: CP Command and Utility Reference
SC24-5776  VM/ESA: CMS Command Reference
SC24-5770  VM/ESA: REXX/VM Reference
SC24-5778  VM/ESA: CMS Pipelines Reference
SC24-5780  VM/ESA: XEDIT Command and Macro Reference
?
q
```

Figure 16. Example of Using the ed Editor

There is also a sed editor. This is a *noninteractive* editor where you enter the sed command, specifying a file containing editing commands and a data file, and it produces an edited target file with no user interaction.

For more information on ed, see *IBM OpenEdition for VM/ESA: Command Reference*, SC24-5728.

vi Editor

The vi editor is a *screen* editor, rather than a line editor. This means that it shows you as much of the file as it can fit on your screen. The name vi is short for visual. It was specifically designed for program editing. This editor can be in one of two states: *command mode* or *input mode*.

The vi editor is not available in OpenEdition for VM/ESA. See 2.9.2, “XEDIT Text Editor” for information on the supported editor.

emacs Editor

The emacs is a macro-driven editor using a single input mode. The name emacs comes from *editor macros*. emacs uses control sequences to manipulate text, For example, Ctrl-K deletes to the end of line.

There is no OpenEdition Shell emacs editor. See 2.9.2, “XEDIT Text Editor” for information on XEDIT.

2.9.2 XEDIT Text Editor

XEDIT is a powerful full-screen editor, and it is easy to master. For readers that wish a short overview, listed are some highlights of this useful editor:

- User-friendly cursor movement keys (left, right, up, and down).
- The ability to tailor all function keys to your particular needs.
- Extended string search facilities for improved text processing.
- Automatic wrapping of lines that are longer than a screen line.
- The ability to directly enter selected subcommands on a displayed line.
- The full-screen layout can be tailored to any format you prefer.
- The physical screen can be divided in multiple views of the same, or of different, files.
- A variety of macros for improved text processing are available, such as macros to join and split lines, macros to display selected file lines, and a macro to sort the contents of your file.
- A HELP facility is available, which provides online full-screen display for all XEDIT subcommands and macros during an editing session.
- Prefix area subcommands that let you easily work with blocks of text.

- A very powerful macro language that lets you customize almost any aspect of your editing session.

Figure 17 shows how a standard XEDIT session appears.

```
 vmgreat.books  V 80  Trunc=80 Size=5 Line=0 Col=1 Alt=0


===== * * * Top of File * * *
      |...+....1....+....2....+....3....+....4....+....5....+....6....+....7...
===== SC24-5773  VM/ESA: CP Command and Utility Reference
===== SC24-5776  VM/ESA: CMS Command Reference
===== SC24-5770  VM/ESA: REXX/VM Reference
===== SC24-5778  VM/ESA: CMS Pipelines Reference
===== SC24-5780  VM/ESA: XEDIT Command and Macro Reference
===== * * * End of File * * *


====>
                                                          X E D I T  1 File
```

Figure 17. Example of a Standard XEDIT Session

Figure 18 shows how a customized XEDIT session appears.

The customized session has a split screen (presenting three different files at the same time) and other modifications.

```
 vmgreat.books  V 80  Trunc=80 Size=5 Li FAVORITE PEOPLE   A1  F 80  Trunc=80 S
====>                                    ====>
* * * Top of File * * *           00000 * * * Top of File * * *           00000
T   T    T    T    T    T    T          Name      Location           Note 00001
SC24-5773  VM/ESA: CP Command and 00001 --------- ------------------ ---- 00002
SC24-5776  VM/ESA: CMS Command Re 00002 Samantha  Kingston     - NY  Adr= 00003
SC24-5770  VM/ESA: REXX/VM Refere 00003 Miguel    Easter Island- NY  Adr= 00004
SC24-5778  VM/ESA: CMS Pipelines  00004 Amparo    Newburgh     - NY  Adr= 00005
|...+....1....+....2....+....3...       Cynthia   Bridgeport   - CT  Adr= 00006
SC24-5780  VM/ESA: XEDIT Command  00005 Linda     Poughkeepsie - NY  Adr= 00007
* * * End of File * * *           00006 Donovan   Endicott     - NY  Adr= 00008
                                       * * * End of File * * *           00009
                                       |...+....1....+....2....+....3...
 1= HELP      2= WSDICT    3= QUIT       1= HELP      2= WSDICT    3= QUIT
 7= BACKWARD  8= FORWARD   9= =          7= BACKWARD  8= FORWARD   9= =
 SCOMDIR  NAMES    S2  V 255  Trunc=255 Size=26 Line=0 Col=1 Alt=0
====>
00000 * * * Top of File * * *
00001 ****************************************************************
00002 *  COPYRIGHT -
00003 *
00004 *    THIS MODULE IS "RESTRICTED MATERIALS OF IBM"
00005 *    5684-112 (C) COPYRIGHT IBM CORP. - 1991
00006 *    LICENSED MATERIALS - PROPERTY OF IBM
00007 *    SEE COPYRIGHT INSTRUCTIONS, G120-2083
00008 *    ALL RIGHTS RESERVED.
      |...+....1....+....2....+....3....+....4....+....5....+....6....+....7...
 1= HELP      2= WSDICT    3= QUIT      4= TABKEY    5= SCHANGE   6= ?
 7= BACKWARD  8= FORWARD   9= =        10= RIGHT 40 11= SPLTJOIN 12= LEFT  40
```

Figure 18. Example of a Customized XEDIT Session

Special Considerations for the Byte File System

Using XEDIT, you can edit only regular files (not special files, such as FIFO files). To edit, you need read permission for the file and search permission for any intermediate directories. You also need write permission to save changes to the file.

When you create a new file, you must have the appropriate permissions to add a new file to the parent directory. When XEDIT creates a file, it attempts to set the permission bits to rw- rw- rw-; for more information on permission bits, see 2.4, "Understanding File System Permissions" on page 11.

When XEDIT starts an editing session, it reads the entire file into your virtual storage. At the end of the session, it replaces the original file with the edited file.

During an XEDIT session, you can use the following types of commands:

- Scrolling commands
- Line commands
- XEDIT subcommands
- CMS and CP commands
- OPENVM commands

Invoking XEDIT to Edit a BFS File

To use XEDIT to edit a file in CMS, enter:

`xedit` *`pathname`* `(nametype bfs`

In the SHELL you can enter:

`cms 'xedit` *`pathname`* `(nametype bfs noprofile'`

pathname is a fully qualified path name, or if the file is in the principal working directory, the file name and extensions. The parameter `nametype bfs noprofile` tells XEDIT to treat the file as a byte file and ignore your XEDIT profile. You may wish to bypass your existing XEDIT profile until you tailor it for BFS-specific operations.

To save some typing you can create a shell script. A shell script is to UNIX as the EXEC is to VM. For example, you can create an x shell script that might contain just the following line:

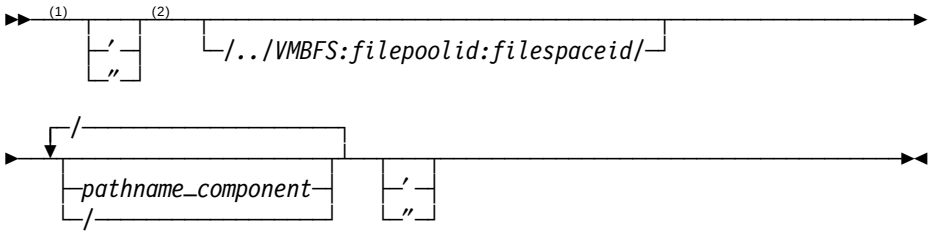
```
cms xedit $1 '(nametype bfs' $2
```

You can also place the following function in `/etc/profile`

```
function xedit {
  cms "XEDIT '$1' (NAMETYPE BFS"
  return
  }
```

All objects in the OpenEdition for VM/ESA Byte File System (BFS) are identified through path names. The path name specified may be relative, absolute, or fully-qualified. If the path name contains special characters such as blanks or parentheses, it must be enclosed in single quotes.

BFS Path Name Syntax:



Notes:

1. Either the /../ **VMBFS:***filepoolid:filespaceid/* or the *pathname_component* indicated in this syntax must be entered.
2. A single quote or a double quote is an optional delimiter, but if specified to signify the start of a path name, it must also be specified to signify the end of the path name.

Two important XEDIT options apply to the BFS: NAMETYPE and BFSLINE.

NAMETYPE XEDIT Option: Since XEDIT is used for both CMS native record files (which use a naming convention of *filename filetype filemode*) and BFS files (which use path names), you must tell XEDIT which naming convention you are using. Do this by using the NAMETYPE CMS or NAMETYPE BFS option of the XEDIT command.

Hints and Tips

If you have customized your own XEDIT profile macro and you use the LOAD subcommand, be aware that the untokenized, mixed case file ID is passed as a *second argument string* to a PROFILE written in REXX. Your normal PROFILE would try to XEDIT a nonexistent tokenized uppercase file ID.

For example, if you enter the following:

```
XEDIT vmgreat.books (nametype BFS
```

you would get an empty file with fn=VMGREAT. (including the period).

To solve this problem, enter the following:

```
Parse Arg fileid '(' options , pathname
```

Then enter:

```
'LOAD' pathname '(' options
```

(After deciding if this is a BFS file, of course.) Most XEDIT profiles were coded to pass all arguments in upper case, as was the standard before support of mixed case files was introduced.

The XEDIT profile must be in REXX, not in EXEC or EXEC2, when the NAMETYPE BFS option is used. Otherwise, you get a message like this: `DMSREX471E Error 35 running PROFILE XEDIT, line 1: Invalid expression.`

The other option that applies to the BFS is BFSLINE.

BFSLINE XEDIT Option: BFSLINE lets you tell the editor how to translate the byte stream into lines of *records*. The end-of-line character is not displayed while using XEDIT against a BFS file. These are the operands of the BFSLINE option in XEDIT:

NL The new-line character (X'15') will be used to delineate lines when reading or writing a BFS file.

lrecl Indicates the file should be treated as a fixed file, with no interpretation of records based on end-of-line characters. The file is presented as a fixed record format (RECFM=F) file with a logical record length (LRECL) equal to the *lrecl*. No end-of-line characters are removed.

CRLF Indicates that carriage return and line feed characters should be used to delineate lines when reading or writing a BFS file.

/string/ Allows the user to specify 1 to 2 characters that are used to delineate lines when reading or writing a BFS file.

/hexstring/ Specifies a hexadecimal string of 2 to 4 characters that defines the value to be used for BFSLINE.

For more information on these options, see *VM/ESA: XEDIT Command and Macro Reference*, SC24-5780.

The following command uses the *lrecl* with the BFSLINE option to produce the results shown in Figure 19 on page 41.

```
cms 'xedit vmgreat.books (BFSLINE 72 NAMETYPE BFS'
```

As you know, BFS files are not record files, but rather they are just a stream of bytes.

When a file is read into an XEDIT session, everything up to the end-of-line character is interpreted as a line and presented as a *record* in the XEDIT session (the end-of-line character is not displayed while XEDITing the BFS file). You use the BFSLINE option to tell XEDIT how to translate a BFS byte stream into records for editing. The default is BFSLINE NL; and it indicates that the new-line character (X'15') should be used to delineate lines when reading or writing a BFS file.

So remember that when you are porting a file from another system that happens to use another end-of-line character, you *must* use the BFSLINE option to XEDIT it.

```
 vmgreat.books  F 72  Trunc=72 Size=4 Line=0 Col=1 Alt=0


===== * * * Top of File * * *
      |...+....1....+....2....+....3....+....4....+....5....+....6....+....7.>
===== SC24-5773  VM/ESA: CP Command and Utility Reference"SC24-5776  VM/ESA: C
===== MS Command Reference"SC24-5770  VM/ESA: REXX/VM Reference"SC24-5778  VM/
===== ESA: CMS Pipelines Reference"SC24-5780  VM/ESA: XEDIT Command and Macro
===== Reference"
===== * * * End of File * * *


====>
                                                          X E D I T  1 File
```

Figure 19. XEDIT Option BFSLINE

Code Page Considerations

A *code page* for a specific character set determines the graphic character produced for each hexadecimal encoding. The code page used is determined by the programs and national languages being used.

When you edit a BFS file using XEDIT, two code pages may be at work and there is no conversion between them. If you have not customized your keyboard, any left or right square bracket you type will be stored as characters that will not be properly interpreted by the C compiler, shell, or utilities.

Across IBM's EBCDIC code pages, there are thirteen characters (shown in Table 6), whose hexadecimal encoding vary.

Table 6. POSIX Characters Commonly Translated

Character Name	**Symbol**
backslash	\
right brace	}
left brace	{
right square bracket	]
left square bracket	[
circumflex	^
tilde	~
exclamation mark	!
number sign (pound sign)	#
vertical bar	|
dollar sign	$
commercial at sign	@
grave accent	`

For more information on working with special characters, see 2.8.3, "Character Translation" on page 31.

Entering Tabs Using XEDIT

Some shell commands, such as make and awk, might require the use of a <tab> character. When using XEDIT, you cannot directly type a tab character (XEDIT handles only displayable characters). The *IBM OpenEdition for VM/ESA: User's Guide*, SC24-5727 recommends typing a substitution character, and then globally changing it (using alter) to an X'05'.

You might prefer to use the SET INPUT command in CMS to achieve this, so you can see the results. Perform the following steps:

- Type: SET INPUT @ 05
- Begin typing and enter tab characters by typing the @ sign.

Figure 20 shows this example before pressing Enter.

```
 TABS     EXAMPLE  A1  F 80  Trunc=80 Size=1 Line=0 Col=1 Alt=2


===== * * * Top of File * * *
      |...+....1....+....2....+....3....+....4....+....5....+....6....+....7...
===== @1@2@3@4@5
===== * * * End of File * * *


====>
                                                          X E D I T  1 File
```

Figure 20. Tab Characters in XEDIT (before Pressing Enter)

Figure 21 shows the same example after pressing Enter.

```
 TABS     EXAMPLE  A1  F 80  Trunc=80 Size=1 Line=0 Col=1 Alt=2


===== * * * Top of File * * *
      |...+....1....+....2....+....3....+....4....+....5....+....6....+....7...
=====     1    2    3    4    5
===== * * * End of File * * *


====>
                                                          X E D I T  1 File
```

Figure 21. Tab Characters in XEDIT (after Pressing Enter)

Note: Other commands related to tabbing are SET IMAGE, EXPAND and COMPRESS. See *VM/ESA: XEDIT Command and Macro Reference*, SC24-5780 for more information.

Preserving Trailing Blanks in Files

XEDIT removes trailing blanks from lines in files unless the BFSLINE LRECL option is used. This is because XEDIT treats all BFS files as being variable record format.

About Lowercase and Mixed-Case Files

The initial case setting in XEDIT is based on the file type of the file being edited. In BFS, the initial setting will be in mixed-case. To enter characters in uppercase, enter the XEDIT subcommand SET CASE UPPERCASE, or use the Caps Lock key on the keyboard. After entering the command, all new characters entered or previously existing lines that are edited will revert to uppercase.

Note: You can change a complete file just by using the UPPERCAS * command.

2.9.3 Full-Screen CMS and the OpenEdition Shell

Full-screen CMS provides an environment closer to a typical UNIX session than normal CMS does.

In full-screen CMS, commands can be entered on the screen immediately under the command prompt, as opposed to in the input area located at the bottom of the screen. Output greater than a single screen can be scrolled forward and back through the use of tailorable PF keys. A command that is incorrectly entered may be modified, and when the user presses Enter, the corrected command is executed.

Figure 22 shows how the POSIX shell appears when using full-screen CMS. Enter SET FULLSCREEN ON to enter full-screen CMS. The SET FULLSCREEN OFF command returns to line-mode CMS.

You can enter full-screen CMS from within a shell session by prefixing the command with cms.

```
drwxrwxrwx   1 maintsv  system         0 Feb 26 18:34 vmop
 maintsv /home/maintsv/ $
cms q time
TIME IS 16:37:48 EDT SUNDAY 05/19/96
CONNECT= 00:00:52 VIRTCPU= 000:00.66 TOTCPU= 000:00.85
 maintsv /home/maintsv/ $
ls
1                 READ_UDF          del_groups        mbox
ACL_ENABLE_GROUP  RGY_ENABLE_USER   del_users         nis2dce_groups
ACL_ENABLE_USER   SUSP_USER         dfs_enable_users  nis2dce_users
ADD_GROUP         WRITE_GDF         get_all_info      null
ADD_USER          WRITE_UDF         get_info_groups   prime
CR_EMPTY_GDF      a                 get_info_users    pwd2dce
CR_EMPTY_UDF      acl_enable_users  gmgt              rgy_enable_users
DEL_GROUP         add_groups        greet             shell_scripts
DEL_USER          add_users         greet1            string_tree
DFS_ENABLE_USER   bboard            greet2            susp_users
ENVIRONMENT       binop             greet3            teldir
GET_ACCOUNT       cdsli             greet4            timop
GET_ACL           cdslir            greet5            tmp
GET_ACL_INI       cemetary_users    greet5.pax        umgt
GET_GROUP         dce_groups        greet6            umgt.tar
GET_PRINCIPAL     dce_tools         greet7            umgt.tarbin
 maintsv /home/maintsv/ $

PF1=Help      2=Pop_Msg   3=Quit       4=Clear_Top  5=Filelist   6=Retrieve
PF7=Backward  8=Forward   9=Rdrlist   10=Left      11=Right     12=Cmd
====>
16:37:58                                                Executing a command
```

Figure 22. Full-Screen CMS and the OpenEdition Shell

2.10 Printers

This section discusses some of the printing facilities available in UNIX and in OpenEdition for VM/ESA.

A printer is usually shared among users to optimize the use of system resources. After entering a print command, the shell prompt returns and you can resume your work. However, this does not mean that your files are actually being printed. The print command just places a file in a print queue, which is a repository for files waiting to be printed.

The print process can be described as having three stages:

1. *Commands* entered by users that request something to be printed.
2. *Spooling queues* that hold and sequentially handle the print jobs.
3. *Control Program processes* that actually handle the transfer of the files from the spool to the printing device.

2.10.1 UNIX Printing

There are some differences in how printing is implemented between various systems. The following section contains a short discussion of the Berkeley software distribution (BSD) and System V printing implementations.

BSD Printing

There are five major programs in the BSD spooling system:

lpr Copies a file to the spool. As discussed earlier, this command just queues a job rather than printing it.

lprm Deletes jobs from the spool.

lpq Gets a list of print jobs currently in the spooling queue.

lpd This is the printer *daemon*. You can think of it as a *data event monitor* (a task, process, or thread that intermittently awakens to perform some chores and then goes back to sleep).

lpc This command is used for maintenance, status display, and manipulation of the printer queues.

System V Printing

There are three major components in the System V printing system:

User commands

These are commands such as:

1. `lp` to initiate a print request
2. `cancel` to cancel a request for print
3. `lpstat` to display the print queue status

Spooling daemon

`lpsched` manages print requests by sending data to the correct printer.

Administration commands

These include commands as `accept`, `reject`, `enable`, `disable`, `lpadmin`, `lpmove`, and `lpusers`.

2.10.2 VM/ESA OpenEdition Printing

In this section some techniques are discussed that allow VM handle your printing requirements by taking advantage of VM's large-volume printing facilities.

The only print command supported in the OpenEdition for VM/ESA is the `lp` command. The options `-m`, `-o`, `-s`, `-t`, `-w` are currently not implemented. The option `-d` allows you to override the standard destination (as defined by the LPDEST environment variable). For the argument to `-d`, you may specify a `node.user`, a `user`, or a nickname defined in your CMS NAMES file.

Figure 23 shows an example of using the `lp` command to send a file to a user specified in the NAMES file. The NAMES file contains:

```
:nick.GUS       :userid.mindreau :node.wtscpok
```

As the target user ID resides in another node, RSCS was chosen to route the output.

```
lp -d gus vmgreat.books

PRT FILE 0014 SENT TO   RSCS     RDR AS  0001 RECS 0005 CPY  001 A NOHOLD NOKEEP
 mindreau /home/mindreau/ $
 Sent file 0001 (0001) on link WTSCVMXA to WTSCPOK(MINDREAU)
 From WTSCPOK: File (0001) spooled to MINDREAU -- origin TOTVM1(MINDREAU) 04/08
 /96 13:23:09 EDT
 From WTSCVMXA: Sent file 1024 (0001) on link WTSCPOK to WTSCPOK(MINDREAU)
```

Figure 23. Example of lp Shell Command

If you use `lp` without a destination option, it will spool to your user ID's virtual printer and eventually be taken care of by the Control Program and printed on a real, physical line printer. Of course, for this to happen you must have a channel-attached printer on your system started in the print class of your print file. The CP QUERY PRINTER command displays the status of your print queue, and the CP PURGE PRINTER command cancels the print job. For more information on these commands, see *VM/ESA: CP Command and Utility Reference*, SC24-5773.

Most modern printers actually are driven by specialized programs like the Advanced Function Printing (AFP) family of products and may require special handling before a job is printed. Several VM installations already implement specialized EXECs to format and route printing to these page printers. The discussion of this implementation is beyond the scope of this publication. But to take advantage of any installation-specific CMS record file print facility, you should first copy the file from the BFS using a command like the following:

```
openvm getbfs vmgreat.books vmgreat books a
```

Another printing-related command is the `pr` command. It formats a file in paginated form and sends it to standard output. Of course you can send this output to the printer by using a shell pipe, such as:

```
pr vmgreat.books | lp
```

Note: The OpenEdition Shell `print` command is tricky because it normally just displays arguments to the standard output using the same conventions as `echo` (which is considered obsolete). Do not confuse the shell `print` command with the CMS PRINT command.

2.11 OPENVM Commands

OpenEdition for VM/ESA provides a set of commands for the manipulation of data residing in the Byte File System (BFS). With these commands, a user can perform a subset of functions that are available from within the shell environment. These commands work from the standard CMS environment and do not require the Shell and Utilities feature to be installed.

OPENVM commands may be used to perform tasks on files, directories, and other BFS objects. These tasks may be:

- Editing files
- Erasing files
- Renaming/moving files
- Changing file permissions and ownership
- Running POSIX applications

2.11.1 Entering Long OPENVM Commands

In a UNIX environment, commands can be incredibly long and exceed the input area of a 3270 terminal. Because of this, the OPENVM commands may require multiple lines. To enter multiple lines, type OPENVM and press Enter. The command interpreter will send a message to the screen prompting for more input lines. Enter as many lines as needed. The command is ended by entering a null line. Figure 24 is an example of entering multiple lines with the OPENVM command.

```
openvm
DMSWOV2140R Enter operands:(enter a null line to indicate that you are finished)

list /home/rodnash/level1/newlevel/level2/level3/test
(name loc h

Null line entered here.

Directory = '/home/rodnash/level1/newlevel/level2/level3/'
File name File type Byte file system  Type  Path name component
5C7       0         VMSYS:ROOT.        F    'test'
Ready; T=0.01/0.02 08:45:15
```

Figure 24. Example of OPENVM Command

2.11.2 OPENVM RUN Command

The OPENVM RUN command allows applications that reside in either a BFS or CMS record file system to be executed. These applications are POSIX applications, and the OPENVM RUN command sets up the environment needed to let these POSIX applications run. The BFS is searched first, and if no match is found, then the CMS file system is searched. Case sensitivity is respected even when the command is interpreted as a CMS file name. Figure 25 on page 47 is an example of a session showing various OPENVM commands.

```
openvm mount /../VMBFS:VMSYS:ROOT/ /                    1
Ready; T=0.01/0.02 09:12:16

openvm set dir /../VMBFS:VMSYS:ROOT/home/rodnash        2
Ready; T=0.01/0.01 09:12:23

openvm listfile                                          3
Directory = '/home/rodnash'
Update-Dt  Update-Tm Type  Links          Bytes Path name component
04/15/1996 16:57:17   F        1            588 '.profile'
04/18/1996 18:58:49   F        1           2155 '.sh_history'
04/15/1996 17:49:05   F        1           3408 'hellorr'
04/12/1996 14:35:34   F        1            250 'count'
04/11/1996 14:00:53   F        1             12 'fred'
04/02/1996 18:22:56   D        -              - 'level1'
04/10/1996 16:52:07   F        1              0 'ls'
04/10/1996 16:17:46   F        1            122 'mik'
04/04/1996 19:14:37   F        1           2263 'results'
04/15/1996 16:22:19   L        -              - 'stage1'
04/10/1996 16:14:21   F        1            125 'testopt'
04/02/1996 12:23:17   F        1             15 'test1'
04/04/1996 14:07:20   F        1             15 'test2'
04/04/1996 14:07:24   F        1             15 'test3'
04/10/1996 14:28:40   F        1             15 'test4'
Ready; T=0.12/0.15 09:12:34

openvm run hellorr                                       4

        ┌──────────────────────────────────────────┐
        │                                          │
        │            Hello - Redbook Readers       │
        │                                          │
        └──────────────────────────────────────────┘
Ready; T=0.02/0.02 09:12:47

listfile hello * * (1                                    5
FILENAME FILETYPE FM FORMAT LRECL       RECS     BLOCKS   DATE     TIME    LABEL
HELLO    MODULE   A1 V       3192          3          1  4/15/96 17:48:14
Ready; T=0.03/0.04 09:13:57

openvm run HELLO                                         6

        ┌──────────────────────────────────────────┐
        │                                          │
        │            Hello - Redbook Readers       │
        │                                          │
        └──────────────────────────────────────────┘
Ready; T=0.01/0.01 09:16:32
```

Figure 25. Example of OPENVM RUN

The execution steps are explained as follows:

1. Mount the root directory of the root file space from the VMSYS file pool server.
2. Set the current working directory to where the program is.
3. List the contents of the directory to make sure the program exists.
4. Run the program.
5. List the CMS program to make sure it exists.
6. Run the CMS POSIX application.

2.11.3 OPENVM GETBFS Command

The OPENVM GETBFS command copies a BFS file into another BFS file, an SFS directory, or a CMS minidisk. The options for this control the way the data is split into records when a BFS file is migrated to the CMS file system. In the sections that follow, you will learn how to manipulate the three most common forms of data in the BFS, namely executable, text, and binary data.

Using GETBFS for Executable Code

When a BFS executable file is copied from the BFS to either a CMS minidisk or an SFS directory, we recommend combining both the following options on the OPENVM GETBFS command:

- Use MODULE option of the OPENVM command.
- The file type of the file ID set to MODULE.

The resulting CMS file will be in the format of a file created by the GENMOD command. Figure 26 is an example of how to use the OPENVM GETBFS command to copy an executable file.

```
openvm query mount                                          1
Mount point = '/'
Type  Stat  Mounted
BFS   R/W  '/../VMBFS:VMSYS:ROOT/'
Ready; T=0.01/0.01 09:32:10

openvm getbfs /home/rodnash/hellorr hello module a          2
Ready; T=0.02/0.02 09:33:04

hello                                                       3
        ┌──────────────────────────────────────────┐
        │                                          │
        │            Hello - Redbook Readers       │
        │                                          │
        └──────────────────────────────────────────┘
Ready; T=0.09/0.11 09:33:07

openvm getbfs /home/rodnash/hellorr hiya file a (module     4
Ready; T=0.02/0.02 09:34:07

rename hiya file a hiya module a                            5
Ready; T=0.01/0.01 09:34:24

hiya                                                        6
        ┌──────────────────────────────────────────┐
        │                                          │
        │            Hello - Redbook Readers       │
        │                                          │
        └──────────────────────────────────────────┘
```

Figure 26. OPENVM GETBFS Command (Module Example)

The execution steps are explained as follows:

1. Check that the BFS file system is mounted.
2. Issue OPENVM GETBFS with a file type of MODULE.
3. Execute HELLO MODULE.
4. Issue OPENVM GETBFS with MODULE option.
5. Rename the file.

6 Execute HIYA MODULE.

Using GETBFS for Text

To retrieve a flat file, such as text, a program listing, or program source, the default option, BFSLINE NL, is all that is needed. Figure 27 is an example of how to use the OPENVM GETBFS command to copy a text file.

```
openvm list /etc/samples/comics.lst                          1
Directory = '/etc/samples/'
Update-Dt  Update-Tm Type  Links          Bytes Path name component
09/14/1995 10:37:20   F        1           1905 'comics.lst'
Ready; T=0.01/0.02 14:16:08

openvm getbfs /etc/samples/comics.lst comics list a          2
Ready; T=0.02/0.02 14:16:40

type comics list a                                           3

Detective Comics:572:Mar:1987:$1.75
Demon:2:Feb:1987:$1.00
Ex-Mutants:1:Sep:1986:$2.60
Justice League of America:259:Feb:1987:$1.00
Boris the Bear:1:Sep:1986:$1.50
Flaming Carrot:14:Oct:1986:$2.75
Demon:4:Apr:1987:$1.00
⋮
```

Figure 27. OPENVM GETBFS Command (Text Example)

The execution steps are explained as follows:

1 Check that the BFS file exists.

2 Issue OPENVM GETBFS, and use the default options to create a newline for each X'15' found in the file.

3 Type the contents of the file just loaded to the CMS file system.

Using GETBFS for Binary Data

To retrieve a non-executable binary file, such as a `.pax` or `.gif` file, use the option (BFSLINE 1 on the GETBFS command. This will create a fixed format file of LRECL 1, and will prevent the addition of extra blanks at the end of the file. Figure 28 on page 50 is an example of how to use the OPENVM GETBFS command to copy a binary file.

```
pax -w ./fred >fred.pax                                    1
 maintsv /home/ $

pax -vf fred.pax
drwxrwxrwx   1 root     system         0  Apr 24 14:42 ./fred/
-rwxrwxrwx   1 rodnash  system      3578  Apr 24 13:48 ./fred/hellorr
-rw-rw-rw-   1 rodnash  system        84  Apr 24 14:46 ./fred/lastcommand
-rwxrwxrwx   1 rodnash  system      3578  Apr 24 13:30 ./fred/rr
 maintsv /home/ $

exit                                                       2
Ready; T=1.56/1.96 15:25:22

openvm getbfs /home/fred.pax fred pax a ( bfsline 1        3
Ready; T=0.15/0.15 15:25:46

openvm erase /home/fred.pax                                4
Ready; T=0.01/0.01 15:26:11

openvm putbfs fred pax a /home/fred.pax ( bfsline none     5
Ready; T=0.07/0.07 15:26:42
```

Figure 28. OPENVM GETBFS Command (Binary Example)

The execution steps are explained as follows:

1. Create and verify the pax file. In this case, the file is an archive of a user directory in the BFS.
2. Exit the shell.
3. Enter the OPENVM GETBFS command to retrieve the BFS data. In this case a BFSLINE of 1 was selected to prevent padding of records at the end of the file.
4. The OPENVM ERASE command removes the archive in the BFS.
5. The OPENVM PUTBFS command recreates the **.**pax file in the BFS.

2.11.4 OPENVM CREATE EXTLINK Command

The OPENVM CREATE EXTLINK command has a very useful option to allow an external link from the shell environment back to the CMS environment for execution of a module. Any existing compiled program with a file type of MODULE can be executed from within the shell when defined by an external link. This allows a shell user to have access to POSIX applications generated as CMS modules, or existing CMS applications that reside on a CMS minidisk or an accessed SFS directory.

For information on the different types of external links, see section 4.9.1, “External Links” on page 99.

Figure 29 on page 51 is an example of using OPENVM CREATE EXTLINK to allow the execution of a CMS module:

```
openvm list /home/rodnash/hellorr                                    1
Directory = '/home/rodnash'
Update-Dt  Update-Tm Type  Links          Bytes Path name component
04/24/1996 09:43:17   F        1           3578 'hellorr'
Ready; T=0.02/0.03 14:31:41

openvm getbfs hellorr HRR MODULE A                                   2
Ready; T=0.02/0.02 14:32:26

hrr                                                                  3

        ┌──────────────────────────────────────────────┐
        │                                              │
        │            Hello - Redbook Readers           │
        │                                              │
        └──────────────────────────────────────────────┘
Ready; T=0.09/0.11 14:32:31

openvm create extlink /home/rodnash/rrr CMSEXEC HRR MODULE A         4
Ready; T=0.01/0.01 14:35:14

openvm shell                                                         5
IBM
Licensed Material - Property of IBM
5654-030 (C) Copyright IBM Corp. 1995
⋮

Monday April 29 02:35:23 PM EDT 1996
 rodnash /home/rodnash/ $

ls -l rrr                                                            6
Erwxrwxrwx   1 rodnash  system        12 Apr 29 14:35 rrr -> HRR MODULE A
 rodnash /home/rodnash/ $

rrr                                                                  7

        ┌──────────────────────────────────────────────┐
        │                                              │
        │            Hello - Redbook Readers           │
        │                                              │
        └──────────────────────────────────────────────┘
 rodnash /home/rodnash/ $
```

Figure 29. OPENVM CREATE EXTLINK Command Example

The execution steps are explained as follows:

1. Check which file to copy back into CMS.
2. Issue OPENVM GETBFS command to copy back into CMS.
3. Check that the newly copied file executes in the CMS environment.
4. Create an external link to HRR MODULE A.
5. Enter POSIX shell environment.
6. Display link to HRR MODULE from rrr.
7. Execute rrr.
8. Exit shell environment.

Note: In this example, an executable file from the shell environment was exported to CMS. Any CMS program with a file type of MODULE can be used for the external link and executed. However, to work best in a POSIX environment, the module should be POSIX-compliant.

2.11.5 Limitations of OPENVM Commands

The OPENVM commands are not meant as a full replacement of the POSIX shell. There are certain functions that cannot be performed using the OPENVM commands, that can be performed in the shell environment. They simply allow administration of the environment.

Process Restrictions

For example, an application or command can be started using the OPENVM RUN command, but it cannot be made a background process, and it cannot be canceled by POSIX commands such as `kill` (of course, a CMS hx command will work).

OPENVM LIST Command Scope

The shell environment has a different view of the BFS than CMS does. The shell sees the BFS as a single entity, even though it may consist of many external or symbolic links spread across multiple servers and even VM/ESA systems.

As an example of this, in the OpenEdition Shell environment, the `ls` command can be used to list all files in a directory structure. To follow all subdirectories, the `-r` option is used. The `ls -ri` command is used to obtain a list of all files within the file system structure and their inode numbers.

If you wish to list the entire contents of a BFS, using the OPENVM LIST command, then each mount point within the BFS must be listed individually. With the default installation setup of `/root`, `/tmp`, and `/var` all being separate mount points, the following three OPENVM commands would have to be entered to achieve the same results as `ls -ri`:

```
OPENVM LIST /../VMBFS:VMSYS:ROOT/ (NAME LOC SUBD H
OPENVM LIST /../VMBFS:VMSYS:TMP/ (NAME LOC SUBD H
OPENVM LIST /../VMBFS:VMSYS:VAR/ (NAME LOC SUBD H
```

If the BFS were set up so that each user owns a file space that is an additional mount point, to get a complete view of the BFS, an OPENVM LIST command would have to be entered for every user.

2.11.6 OPENVM Command Cross Reference

Table 7 provides a cross reference between CMS OPENVM commands and POSIX shell command functions.

Table 7 (Page 1 of 2). CMS and Shell Command Equivalents

CMS	Shell	Function
openvm create directory	mkdir	Create a directory. The mkdir command has the option `-p` for creating intermediate directories if they do not already exist.
openvm create extlink	-	Create a BFS path name to be used to reference a file or other data that resides outside the BFS.
openvm create link	ln	Link another name to a file, in addition to its original name.
openvm create symlink	ln	Create a BFS path name to be used to reference an object residing in one BFS using a path name in the same or another BFS.

Table 7 (Page 2 of 2). CMS and Shell Command Equivalents		
CMS	**Shell**	**Function**
openvm erase	rm	Remove, erase, or delete a file from a directory.
	rmdir	Remove, erase, or delete a directory that is empty of files.
openvm getbfs	cp	Copy a file from the BFS to the CMS file system.
openvm listfile	ls	List the files in a directory.
openvm mount	-	Add a BFS or part of a BFS directory tree to the file hierarchy.
openvm owner	chgrp	Change the group owner of a file or directory.
	chown	Change the owner or group of a file or directory.
openvm parchive	pax	Process archive files. Either create a file and directory archive or expand a file and directory archive.
openvm permit	chmod	Change access permission to a file or directory. To use this command, the user must have the appropriate privileges.
openvm putbfs	cp	Copy a file from the CMS file system to the BFS.
openvm query directory	pwd	Display the current working directory.
openvm query link	ls -l	Display the information associated with a symbolic or external link.
openvm query mask	umask	Display file creation mask values.
openvm query mount	-	Display mounted BFS hierarchy.
openvm rename	mv	Move a file from one directory to another or rename a file or directory.
openvm run	-	Execute an object that resides in either the BFS or CMS record file system.
openvm set dir	cd	Change the working directory.
openvm set mask	umask	Define the file creation mask used when creating a file.
openvm unmount	-	Remove a previously mounted BFS or BFS subdirectory tree from your hierarchy.

2.12 File Transfer and Archive

This section discusses several methods for migrating files from one POSIX-compliant system to the next, with relevance to OpenEdition for VM/ESA. Subjects include how tape archive (`tar`), copy in or out (`cpio`), and portable archive exchange (`pax`) work in UNIX and how they are implemented in OpenEdition for VM/ESA.

Although File Transfer Program (FTP, part of the TCP/IP licensed product) is not part of OpenEdition for VM/ESA, it is discussed here, as it is a fast and reliable way to move data from UNIX into VM.

2.12.1 File Transfer Program

The File Transfer Program (FTP) allows you to transfer data and copy files between two computers. Both systems do not need to be running the same operating system, but they do need to be connected by a network that FTP can use.

FTP creates a command interpreter similar to UNIX, but do not expect the shell you are used to. Instead, FTP displays its own prompt messages and uses its special command set for the transfer of files.

Even if you do not know anything about UNIX, you might already have used FTP.

On VM/ESA, FTP is part of the TCP/IP program product. It allows access of CMS record data (SFS or conventional). BFS data must be moved from the BFS into a CMS record file. If FTP is configured to do so, FTP allows you to login with your user ID and password to a VM system, and access the data on any minidisk your virtual machine has access to (established in the CP directory or through ESM privileges). Figure 30 shows how to establish an FTP session from VM to a UNIX system.

```
ftp risc01
VM TCP/IP FTP V2R2
Connecting to RISC01 9.12.14.200, port 21
220 risc01 FTP server (Version 4.1 Mon Aug 21 10:34:44 CDT 1995) ready.
USER (identify yourself to the host):
root
>>>USER root
331 Password required for root.
Password:
>>>PASS ********
230 User root logged in.
Command:
help
User-FTP understands these commands:
  ?        acct      append    ascii    binary     cd      close
  cms      debug     delete    delimit  dir        ebcdic  euckanji
  get      help      ibmkanji  jis78kj  jis83kj    lcd     locsite
  locstat  lpwd      ls        mdelete  mget       mode    mput
  noop     open      pass      put      pwd        quit    quote
  rename   sendport  sendsite  site     sjiskanji  status  struct
  sunique  system    type      user
Specify a command by any unambiguous prefix
Specify a local file by name.type.mode or name.type
Default mode is * for PUT and a current local directory for MPUT, MGET and GET
For information about a particular command, say "HELP command"
Command:
```

Figure 30. FTP Logon and Help Example

The following is a short description of some available commands:

acct Use the acct subcommand to send host-dependent account information.

append Use the append subcommand to append a local file to a remote file. To append to a file on the remote host, you must have a working directory defined (see CD), and you must have write privileges for it.

ascii Use the ascii subcommand to change the file transfer type to ASCII. This is the default transfer type.

binary Use the binary subcommand to change the file transfer type to image. You can also specify if files are stored either in fixed or in variable record format. The latter is the default.

cd Use the cd subcommand to change the remote working directory or file group.

close Use the close subcommand to disconnect from the remote host.

cms Use the cms command prefix to pass a command to the local CMS environment (do not use synonyms).

debug Use the debug subcommand to toggle the internal debug options.

delete Use the delete subcommand to erase a specified foreign host file.

delimit Use the delimit subcommand to change the delimiter between file name and file type.

dir Use the dir subcommand to list the directory entry of a remote set of files, file group, or directory. Directory entries listed are complete, giving auxiliary information about the file. For a list of just the file names in a directory, use the ls subcommand.

ebcdic Use the ebcdic subcommand to change the file transfer type to EBCDIC.

get Use the get subcommand to copy a remote file into a local file with name *localname*. To get a file from the remote host, you must have a working directory defined (see cd), and you must have read privileges for it. If a file with the name *localname* already exists and the replace option is not used, then the old file is not overwritten and a message is given.

help Use the help subcommand to assist you in using FTP. If invoked with the parameter SERVER, then whatever help the remote host offers is what you get.

Figure 31 shows the output of HELP SERVER in this system.

```
help server
>>>HELP
The foreign server has this help:
214- The following commands are recognized (* =>'s unimplemented).
   USER    PORT    STOR    MSAM*   RNTO    NLST    MKD     CDUP
   PASS    PASV    APPE    MRSQ*   ABOR    SITE    XMKD    XCUP
   ACCT*   TYPE    MLFL*   MRCP*   DELE    SYST    RMD     STOU
   SMNT*   STRU    MAIL*   ALLO    CWD     STAT    XRMD    SIZE
   REIN    MODE    MSND*   REST    XCWD    HELP    PWD     MDTM
   QUIT    RETR    MSOM*   RNFR    LIST    NOOP    XPWD
214 Direct comments to ftp-bugs@risc01.
Command:
```

Figure 31. HELP SERVER Example

locstat Use the locstat subcommand to display local status information.

ls Use the ls subcommand to list only the names of the files in a remote set of files, file group, or directory. Only the file names are listed by the ls subcommand. (For a list of complete directory entries, giving auxiliary information about the file, use the dir subcommand).

mdelete Use the mdelete subcommand to erase a foreign file.

mget Use the mget subcommand to copy multiple foreign files into local files.

mode Use the mode subcommand to specify what mode, or data format, the file transfer is to take place in. mode defines how the bits of the data are transmitted. mode has two parameters:

- **B** Refers to BLOCK mode. When using BLOCK mode, data is transmitted as a series of data blocks, preceded by one or more header bytes. BLOCK allows the transfer of binary data, while preserving the logical record length (LRECL) of the file.
- **S** Refers to STREAM mode. When using this mode, data is transmitted as a stream of bytes. Any representation type may be used with STREAM mode (see the TYPE subcommand). STREAM mode is the default transfer mode.

For a VM-to-VM transfer, specify MODE BLOCK and EBCDIC. This will maintain the actual record boundaries through the transfer.

mput Use the mput subcommand to send files to the foreign host.

noop Use the noop subcommand to see if the remote host is still responding.

open Use the open subcommand to connect to a remote host.

pass Use the pass subcommand to send your password to the remote host.

put Use the put subcommand to copy a local file to the foreign host.

pwd Use the pwd subcommand to print the name of the remote working directory or file group.

quit Use the quit subcommand to exit FTP and disconnect from the remote host.

quote Use the quote subcommand to send an uninterpreted string to the server port on the remote host.

quote bypasses the local user FTP interface. It may be used to send a command to the server FTP that the user FTP does not recognize.

rename Use the rename subcommand to rename a file on the remote host.

sendport Use the sendport subcommand to toggle the PORT commands.

sendsite Use the sendsite subcommand to toggle the usage of the SITE command.

site Use the site subcommand to send site-dependent information essential for file transfer. Many remote hosts do not require such information.

status Use the status subcommand to check the connection status of the remote FTP.

struct Use the struct subcommand to specify the structure of the file. With a file structure, the file is considered to be a continuous sequence of data bytes. This is the only structure supported.

sunique Use the sunique subcommand to toggle the usage of the stou command. By default, FTP disables the usage of the stou command when copying a local file to a foreign host with the put or mput

subcommand. Here, the stor command is used and foreign files that already exist are overwritten.

system Use the system subcommand to query the type of the remote operating system.

type Use the type subcommand to specify what representation, or type, the file transfer is to take place in.

I Refers to IMAGE type. When using IMAGE type, data is sent as contiguous bits, packed into eight bit bytes. It is intended for the efficient storage and retrieval of files and for the transfer of binary data.

A Specifies the transfer type as ASCII, the default transfer type. This has the same effect as the ascii subcommand. No vertical format information is transferred. It is intended primarily for the transfer of the text files, except when both hosts would find the EBCDIC type more convenient.

E Specifies the transfer type as EBCDIC. This has the same effect as the ebcdic subcommand. No vertical format information is transferred. EBCDIC transfer type is intended for an efficient transfer of files between hosts that use EBCDIC for their internal character representation.

user Use the user subcommand to identify yourself to the remote host.

2.12.2 TAR

The tar command is a method to archive files. An *archive* is a single file that contains the complete contents of a set of other files; an archive preserves the directory hierarchy that contained the original files.

This command is useful when an elaborate backup process is not needed. For a production system backup, the recommended procedure is a complete BFS backup (for example: FILEPOOL BACKUP). For backing up a small list of files, the tar command may be appropriate.

One of the advantages of tar is that it moves files in a directory tree. This is often used as a way to distribute software, in particular over *anonymous ftp*.

Figure 32 on page 58 shows an example using tar. (Normally, you do not want to write the backup file to your own directory, but we are doing this for demonstrative purposes.) For more information on tar, see *IBM OpenEdition for VM/ESA: Command Reference*, SC24-5728.

```
tar -cf backup/backup.1 /home/mindreau
GSU7227 tar: /home/mindreau/backup/backup.1: grew in size
 mindreau /home/mindreau/ $
tar -tf backup/backup.1
tar: blocksize = 20
/home/mindreau/
/home/mindreau/.sh_history
/home/mindreau/posix.terms
/home/mindreau/sample.c
/home/mindreau/sample.o
/home/mindreau/test.test
/home/mindreau/vmgreat.books
/home/mindreau/backup/
/home/mindreau/backup/backup.1
/home/mindreau/examples/
/home/mindreau/examples/sample.1
 mindreau /home/mindreau/ $
```

Figure 32. tar Utilization Example

For an example of restoring a tar file from another system, see 2.12.4, “PAX” on page 59.

2.12.3 CPIO

The cpio command copies file archives from *in* to *out*. The main advantage of cpio is that it copies files much faster than tar does. Furthermore, cpio can backup files that describe devices.

A cpio archive is a concatenation of files and directories preceded by a header giving the file name and other file system information.

cpio takes a list of files from stdin and writes the archive to stdout. This function is unique compared with tar and pax, and is often used within a shell pipe.

With cpio you can create a new archive, extract contents of an existing archive, list archive contents, and copy files from one directory to another.

Figure 33 on page 59 shows an example using cpio. Normally when you use absolute path names when creating an archive file, the files in the archive are restored to the same position in the hierarchy, regardless of what directory you are working in when you restore the files. In the example, a relative path name to restore the files to the *backup* directory is used. For more information on cpio, see *IBM OpenEdition for VM/ESA: Command Reference*, SC24-5728.

```
ls vmgreat.books | cpio -o > cpio.bkup
1 block
 mindreau /home/mindreau/ $
cd backup
 mindreau /home/mindreau/backup/ $
cpio -i < ../cpio.bkup
1 block
 mindreau /home/mindreau/backup/ $
ls -l
total 240
-rw-rw-rw-   1 mindreau system     86016 Apr 29 14:56 backup.1
-rw-rw-rw-   1 mindreau system     31744 Apr 29 17:37 cpiobkup.1
-rw-rw-rw-   1 mindreau system       226 Apr 30 13:40 vmgreat.books
 mindreau /home/mindreau/backup/ $
```

Figure 33. cpio Utilization Example

2.12.4 PAX

pax (the UNIX command for the *portable archive interchange*) reads and writes archive files. The pax command combines the power of the two popular commands tar and cpio.

pax can read and write files in cpio ASCII format, cpio binary format, or tar format. An *archive file* concatenates the contents of files and directories, and it can also record information such as file modification dates, owner names, and so on. You can use a single archive file to transfer a directory structure from one machine to another.

A file stored inside an archive is called a *component file*; similarly, a directory stored inside an archive is called a *component directory*. Together, component files and directories make up the *components* of the archive file.

The best way to appreciate pax is with an example. In this example, data will be moved from a production UNIX system to OpenEdition for VM/ESA. cpio, tar and pax will be used to backup the data, and pax will restore all three formats.

You begin this operation with a *login* to the foreign system. Figure 34 on page 60 shows how to log in using TELNET.

```
telnet risc01
VM TCP/IP Telnet V2R2
Connecting to RISC01 9.12.14.200, port TELNET (23)

Using Line Mode...

Notes on using TELNET when in Line Mode:
- To hide Password, Hit PF3 or PF15
- To enter Telnet Command, Hit PF4-12, or PF16-24
telnet (risc01)

AIX Version 4
(C) Copyrights by IBM and by others 1982, 1994.

login: root
root's Password:
Last unsuccessful login: Wed Mar 20 14:35:59 EST 1996 on /dev/lft0
Last login: Tue Apr 30 11:53:43 EDT 1996 on /dev/pts/4 from wtscpok.itso.ibm.com

#
```

Figure 34. TELNET Login to Remote UNIX Host

Figure 35 shows the steps to backup one file, a directory, and the complete tree belonging to a particular user.

```
# cd /home/mindreau
# ls -al
total 10
-rwxr-----   1 root     system       254 Apr 30 14:33 .profile
-rw-r--r--   1 root     system        11 Apr 30 14:35 file-sample.1
-rw-r--r--   1 root     system        11 Apr 30 14:35 file-sample.2
-rw-r--r--   1 root     system        11 Apr 30 14:35 file-sample.3
-rw-r--r--   1 root     system        11 Apr 30 14:35 file-sample.4
-rw-r--r--   1 root     system        11 Apr 30 14:35 file-sample.5
-rw-r--r--   1 root     system        11 Apr 30 14:35 file-sample.6
drwxr-xr-x   2 root     system       512 Apr 30 14:34 subdir1
drwxr-xr-x   2 root     system       512 Apr 30 14:34 subdir2
drwxr-xr-x   2 root     system       512 Apr 30 14:34 subdir3
# ls file-sample.1 | cpio -oc > backup.cpio
1 blocks
# tar -cf backup.tar subdir1
#
# cd /home
# ls -l
total 11
-rw-r--r--   1 root     system       265 Mar 12 10:29 env3151
-rwxr-xr-x   1 root     system       297 Mar 12 10:30 envXst
drwxr-xr-x   2 guest    usr          512 Jan 04 12:43 guest
drwxr-xr-x   2 hilding  staff        512 Mar 14 09:52 hilding
drwxr-xr-x   4 root     system       512 Mar 04 15:56 jfadel
drwx------   2 root     system       512 Jan 04 12:43 lost+found
drwxr-xr-x   5 root     system       512 Apr 30 14:42 mindreau
drwxr-xr-x   2 newling  staff        512 Feb 19 14:52 newling
drwxrwxrwx   3 rconway  staff        512 Feb 07 17:15 rconway
drwxr-xr-x   2 root     system       512 Mar 19 18:02 test
drwxr-xr-x   2 root     system       512 Apr 08 17:46 vetter
# pax -wf mindreau/backup.pax mindreau
```

Figure 35. Backup Examples Using cpio, tar and pax

As a result of the previous commands, you have the *backup.** files shown in Figure 36.

```
# ls -la mindreau
total 137
drwxr-xr-x   5 root     system       512 Apr 30 14:48 .
drwxr-xr-x  11 bin      bin          512 Apr 30 14:32 ..
-rwxr-----   1 root     system       254 Apr 30 14:33 .profile
-rw-r--r--   1 root     system       512 Apr 30 14:38 backup.cpio
-rw-r--r--   1 root     system     51200 Apr 30 14:48 backup.pax
-rw-r--r--   1 root     system     10240 Apr 30 14:42 backup.tar
-rw-r--r--   1 root     system        11 Apr 30 14:35 file-sample.1
-rw-r--r--   1 root     system        11 Apr 30 14:35 file-sample.2
-rw-r--r--   1 root     system        11 Apr 30 14:35 file-sample.3
-rw-r--r--   1 root     system        11 Apr 30 14:35 file-sample.4
-rw-r--r--   1 root     system        11 Apr 30 14:35 file-sample.5
-rw-r--r--   1 root     system        11 Apr 30 14:35 file-sample.6
drwxr-xr-x   2 root     system       512 Apr 30 14:41 subdir1
drwxr-xr-x   2 root     system       512 Apr 30 14:41 subdir2
drwxr-xr-x   2 root     system       512 Apr 30 14:41 subdir3
```

Figure 36. Results of Previous Three Backup Commands in UNIX

The following step migrates the *backup* files to CMS. You can use FTP to move them into your A-disk as shown in Figure 37 on page 62, or into an SFS directory.

```
ftp risc01
VM TCP/IP FTP V2R2
Connecting to RISC01 9.12.14.200, port 21
220 risc01 FTP server (Version 4.1 Mon Aug 21 10:34:44 CDT 1995) ready.
USER (identify yourself to the host):
root
>>>USER root
331 Password required for root.
Password:
>>>PASS ********
230 User root logged in.
Command:
cd /home/mindreau
>>>CWD /home/mindreau
250 CWD command successful.
Command:
dir
>>>PORT 9,12,14,1,7,94
200 PORT command successful.
>>>LIST
150 Opening data connection for /bin/ls.
total 135
-rwxr-----   1 root     system       254 Apr 30 14:33 .profile
-rw-r--r--   1 root     system       512 Apr 30 14:38 backup.cpio
-rw-r--r--   1 root     system     51200 Apr 30 14:48 backup.pax
-rw-r--r--   1 root     system     10240 Apr 30 14:42 backup.tar
⋮
                .
drwxr-xr-x   2 root     system       512 Apr 30 14:41 subdir3
226 Transfer complete.
Command:
binary
>>>TYPE i
200 Type set to I.
Command:
get backup.cpio backup.cpio.a
>>>PORT 9,12,14,1,7,95
200 PORT command successful.
>>>RETR backup.cpio
150 Opening data connection for backup.cpio (512 bytes).
226 Transfer complete.
512 bytes transferred. Transfer rate 6.17 Kbytes/sec.
Command:
get backup.tar  backup.tar.a
>>>PORT 9,12,14,1,7,96
200 PORT command successful.
>>>RETR backup.tar
150 Opening data connection for backup.tar (10240 bytes).
226 Transfer complete.
10240 bytes transferred. Transfer rate 25.22 Kbytes/sec.
Command:
get backup.pax  backup.pax.a
>>>PORT 9,12,14,1,7,97
200 PORT command successful.
>>>RETR backup.pax
150 Opening data connection for backup.pax (51200 bytes).
226 Transfer complete.
51200 bytes transferred. Transfer rate 168.42 Kbytes/sec.
Command:
```

Figure 37. FTP Utilization to Get Backup Files from UNIX

The next step is to place these files into the BFS. As shown in Figure 38, the command OPENVM is used for this purpose, and then the shell is entered.

```
openvm putbfs backup cpio a /home/mindreau/backup.cpio (bfsline none
Ready; T=0.02/0.02 14:58:24
openvm putbfs backup tar  a /home/mindreau/backup.tar  (bfsline none
Ready; T=0.02/0.02 14:58:32
openvm putbfs backup pax  a /home/mindreau/backup.pax  (bfsline none
Ready; T=0.02/0.03 14:58:39
openvm shell

IBM
Licensed Material - Property of IBM
5654-030 (C) Copyright IBM Corp. 1995
(C) Copyright Mortice Kern Systems, Inc., 1985, 1993.
(C) Copyright Software Development Group, University of Waterloo, 1989.

All Rights Reserved.

U.S. Government users - RESTRICTED RIGHTS - Use, Duplication, or
Disclosure restricted by GSA-ADP schedule contract with IBM Corp.

IBM is a registered trademark of the IBM Corp.

Tuesday April 30 02:58:46 PM EDT 1996
 mindreau /home/mindreau/ $
```

Figure 38. OPENVM PUTBFS Example

In the OpenEdition Shell for this example, you find the three newly created archive files (*backup.**), as created by OPENVM PUTBFS. Before restoring them, display their contents to be certain that the right files have been backed up. You can follow this operation in Figure 39 on page 64.

```
ls -l
total 216
drwxrwxrwx   1 mindreau system         0 Apr 30 13:40 backup
-rw-rw-rw-   1 mindreau system       512 Apr 30 14:58 backup.cpio
-rw-rw-rw-   1 mindreau system     51200 Apr 30 14:58 backup.pax
-rw-rw-rw-   1 mindreau system     10240 Apr 30 14:58 backup.tar
drwxrwxrwx   1 mindreau system         0 Apr 29 14:40 examples
-rw-r-----   1 mindreau staff      22022 Apr 22 14:26 posix.terms
-rw-rw-rw-   1 mindreau system      3160 Apr 29 13:44 sample.c
-rw-rw-rw-   1 mindreau system      6000 Apr 29 13:49 sample.o
-rw-rw-rw-   1 mindreau system         5 Apr 16 14:19 test.test
-rw-rw-rw-   1 mindreau system       226 Apr  4 15:25 vmgreat.books
drwxrwxrwx   1 mindreau system         0 Apr 30 14:57 z_pax-restore
 mindreau /home/mindreau/ $
pax -f backup.cpio
file-sample.1
 mindreau /home/mindreau/ $
pax -f backup.tar
subdir1/
subdir1/subfile-sample.1
 mindreau /home/mindreau/ $
pax -f backup.pax
mindreau
mindreau/.profile
mindreau/subdir1
mindreau/subdir1/subfile-sample.1
mindreau/subdir2
mindreau/subdir2/subfile-sample.2
mindreau/subdir3
mindreau/subdir3/subfile-sample.3
mindreau/file-sample.1
mindreau/file-sample.2
mindreau/file-sample.3
mindreau/file-sample.4
mindreau/file-sample.5
mindreau/file-sample.6
mindreau/backup.cpio
mindreau/backup.tar
mindreau/backup.pax
 mindreau /home/mindreau/ $
```

Figure 39. Displaying the Contents of Your Backup Files

In this section, the files that were previously backed up are restored. Figure 40 on page 65 shows the restore process. The two machines use different code pages. Therefore as you are restoring the files from UNIX to OpenEdition for VM/ESA, you must convert from one character set to another. Fortunately, the shell command pax option -o will take care of this. It converts data represented with the ASCII character set (ISO-8859-1) format into the IBM-1047 character set. Notice the last pax command. This restores the complete tree into the z_pax-restore directory.

Keep in mind that the -o option is global for the entire archive. Every file (component) in the archive will be converted to the code page specified by the -o option. Make sure this is what you really want.

```
pax -rf backup.cpio -o from=iso8859-1,to=ibm-1047
 mindreau /home/mindreau/ $
ls -l
total 224
drwxrwxrwx   1 mindreau system         0 Apr 30 13:40 backup
-rw-rw-rw-   1 mindreau system       512 Apr 30 14:58 backup.cpio
-rw-rw-rw-   1 mindreau system     51200 Apr 30 14:58 backup.pax
-rw-rw-rw-   1 mindreau system     10240 Apr 30 14:58 backup.tar
drwxrwxrwx   1 mindreau system         0 Apr 29 14:40 examples
-rw-r--r--   1 mindreau system        11 Apr 30 14:35 file-sample.1
-rw-r-----   1 mindreau staff      22022 Apr 22 14:26 posix.terms
-rw-rw-rw-   1 mindreau system      3160 Apr 29 13:44 sample.c
-rw-rw-rw-   1 mindreau system      6000 Apr 29 13:49 sample.o
-rw-rw-rw-   1 mindreau system         5 Apr 16 14:19 test.test
-rw-rw-rw-   1 mindreau system       226 Apr  4 15:25 vmgreat.books
drwxrwxrwx   1 mindreau system         0 Apr 30 14:57 z_pax-restore
 mindreau /home/mindreau/ $
pax -rf backup.tar -o from=iso8859-1,to=ibm-1047
 mindreau /home/mindreau/ $
ls -l
total 224
drwxrwxrwx   1 mindreau system         0 Apr 30 13:40 backup
-rw-rw-rw-   1 mindreau system       512 Apr 30 14:58 backup.cpio
-rw-rw-rw-   1 mindreau system     51200 Apr 30 14:58 backup.pax
-rw-rw-rw-   1 mindreau system     10240 Apr 30 14:58 backup.tar
drwxrwxrwx   1 mindreau system         0 Apr 29 14:40 examples
-rw-r--r--   1 mindreau system        11 Apr 30 14:35 file-sample.1
-rw-r-----   1 mindreau staff      22022 Apr 22 14:26 posix.terms
-rw-rw-rw-   1 mindreau system      3160 Apr 29 13:44 sample.c
-rw-rw-rw-   1 mindreau system      6000 Apr 29 13:49 sample.o
drwxrwxrwx   1 mindreau system         0 Apr 30 14:41 subdir1
-rw-rw-rw-   1 mindreau system         5 Apr 16 14:19 test.test
-rw-rw-rw-   1 mindreau system       226 Apr  4 15:25 vmgreat.books
drwxrwxrwx   1 mindreau system         0 Apr 30 14:57 z_pax-restore
 mindreau /home/mindreau/ $
ls -l subdir1
total 8
-rw-r--r--   1 mindreau system        11 Apr 30 14:41 subfile-sample.1
 mindreau /home/mindreau/ $
cd z_pax-restore
 mindreau /home/mindreau/z_pax-restore/ $
pax -rf ../backup.pax -o from=iso8859-1,to=ibm-1047
 mindreau /home/mindreau/z_pax-restore/ $
ls -l
total 0
drwxrwxrwx   1 mindreau system         0 Apr 30 14:48 mindreau
 mindreau /home/mindreau/z_pax-restore/ $
ls -l mindreau
total 112
-rw-r--r--   1 mindreau system       512 Apr 30 14:38 backup.cpio
-rw-r--r--   1 mindreau system     20480 Apr 30 14:48 backup.pax
-rw-r--r--   1 mindreau system     10240 Apr 30 14:42 backup.tar
-rw-r--r--   1 mindreau system        11 Apr 30 14:35 file-sample.1
-rw-r--r--   1 mindreau system        11 Apr 30 14:35 file-sample.2
-rw-r--r--   1 mindreau system        11 Apr 30 14:35 file-sample.3
-rw-r--r--   1 mindreau system        11 Apr 30 14:35 file-sample.4
-rw-r--r--   1 mindreau system        11 Apr 30 14:35 file-sample.5
-rw-r--r--   1 mindreau system        11 Apr 30 14:35 file-sample.6
drwxrwxrwx   1 mindreau system         0 Apr 30 14:41 subdir1
drwxrwxrwx   1 mindreau system         0 Apr 30 14:41 subdir2
drwxrwxrwx   1 mindreau system         0 Apr 30 14:41 subdir3
 mindreau /home/mindreau/z_pax-restore/ $
```

Figure 40. Restoring Your Archive Files

2.13 Programming Extensions for OpenEdition

With the OPENVM commands introduced in VM/ESA Version 2 Release 1.0, you are able to manipulate POSIX applications residing in and out of the BFS. However, complex iterative tasks are best suited to a programming language. REXX and CMS Pipelines were enhanced to create a smooth interface to the OpenEdtion environment.

2.13.1 REXX Extensions

A new REXX subcommand environment, ADDRESS OPENVM, is provided to execute the CSL POSIX functions as REXX functions.

Section 6.4, “CSL Calls Specific to UIDs and GIDs” on page 149 contains an example of OPENVM CSL calls in a REXX environment. For more information about application programming concepts, refer to *VM/ESA CMS Application Development Guide*, SC24-5761.

2.13.2 CMS Pipelines Extensions

CMS Pipelines, as shipped with VM/ESA Version 2 Release 1.0, provides device drivers for reading and writing text files with automatic conversion from records to BFS byte stream and vice versa; it also contains device drivers that can access the byte stream directly without assuming the files have any structure. In addition, device drivers are available to query and set attributes of the POSIX process, such as the current working directory and the user file creation mask. These device drivers are briefly described in the following sections.

bfs—Read or Append to a Binary File

When it is first in a pipeline, `bfs` reads the bytes of a file without deblocking at line ends. Each output record contains some unspecified number of bytes from the file.

When it is not first in a pipeline, `bfs` appends the records it reads from the pipeline to the contents of the file without adding line end sequences.

To read a byte stream file from the current working directory in a Byte File System, and separate the file into records with at least a character length of 1, enter the following PIPE command:

```
pipe bfs /u/cal/.profile | deblock string x0D25 | pad 1 | > /u/cal/.profile
```

The DEBLOCK stage command will identify, within the byte stream file, logical records that are separated with X′0D25′.

bfsdirectory—Read a Directory into the Pipeline

`bfsdirectory` writes an output record for each link and symbolic link in the directory file. The record contains the file name or the symbolic link name. The output from `bfsdirectory` is often fed to `bfsstate` to display information about the file.

To display the contents of the root directory, enter the following PIPE command:

```
pipe bfsdirectory /../ | console
```

bfsquery—Write OpenEdition Information into the Pipeline

bfsquery reads queries from its input stream and writes the results of the queries to its output stream. These queries are supported by the following parameters:

PWD Write the current working directory into the pipeline.

SYMLINK Write the contents of a symbolic link into the pipeline. It accepts a pathname as a parameter.

UNAME Write the name of the current operating system into the pipeline.

To display to the terminal the pathname of the alias command contained in a symbolic link, enter the following PIPE command:

```
pipe literal symlink /bin/alias | bfsquery | console
```

bfsreplace—Overwrite a Binary File

The specified file is overwritten with the data read from the input stream. The file is truncated and then overwritten. Note that POSIX requires no support for units of work, commit, and roll back, and no support is provided for BFS files.

To create a simple shell script in the BFS and run it, enter the following commands:

```
pipe literal echo "Hello World!" | bfsreplace myshell.scp
OPENVM SHELL /u/maintsv/myshell.scp
```

It is interesting that even though the file myshell.scp does not have the execution bit set by default, it will run when called as a parmeter of the shell.

bfsstate—Write Information about Files into the Pipeline

bfsstate reads path names from its input and writes information about the named files to its output stream in a format that resembles the output from the UNIX ls command. The information includes the mode bits, the number of links to the file, the owner and group, the file size, the time of last change, and the file name.

```
pipe literal myshell.scp| bfsstate | console
  -rw-rw-rw-   1 maintsv                19 19960716131223 myshell.scp
  Ready; T=0.01/0.01 09:17:27
```

bfsxecute—Issue OpenEdition Commands

bfsexecute reads commands from its input stream and passes the requests to OpenEdition for VM/ESA. The following commands are supported:

CHDIR Changes the current working directory.

CHMOD Changes the mode or permissions of a file or directory.

LINK Links a path to an object. This is a “hard” link; the path and the object must be within the same file system.

MKDIR Creates an empty directory.

MKFIFO Creates a named pipe.

RENAME Replaces a file or directory with another file or directory. “Rename” is a misnomer; if you are accustomed to CMS, the rename function provides a way to replace a file with some other file in an atomic way. Other processes will always see either the old file or the new file; they will never not see any file.

RMDIR Removes an empty directory from its parent directory; deletes the directory file.

SYMLINK Links a path to an object. This is a “soft” link. The path and the object can be in different file systems.

UNLINK Deletes a link to a file.

bfsexecute is a synonym for bfsxecute. The following PIPE executes the chmod command:

```
pipe literal chmod myshell.scp 755 | bfsxecute
```

Be aware that the syntax order for these commands vary from the standard shell commands. In the case above, the chmod option accepts the path as the first parameter, and the mode must be decimal.

2.14 Tools Making Life Easy

The experienced VM user may require an adjustment time before being comfortable in the POSIX shell environment. The lack of familiar tools that have been used in the VM environment for years can be frustrating. However, with a little thought and imagination, it is easy to develop tools that will make the transition easier. The following section describes some tools that you may want to use to make this transition. These tools are available on the included diskette, or by FTP, as described in D.4, “Getting the Redbook Sample Programs from the Net” on page 204.

2.14.1 BFSLIST

BFSLIST was originally developed by a customer ESP site and then enhanced during the writing of this book. BFSLIST is a REXX EXEC that uses Pipelines and XEDIT to give you a FILELIST-like interface to the BFS. It can be used from the CMS environment or from the POSIX shell environment. This tool makes working with the BFS easy.

BFSLIST has the following components:

- BFSLIST EXEC
- BFSLIST XEDIT
- BFSEXAM XEDIT
- BFSDEL XEDIT

Figure 41 on page 69 is an example of the BFSLIST screen:

```
 RODNASH  BFSLIST  A0  V 80  Trunc=80 Size=26 Line=1 Col=1 Alt=0
DIRECTORY: ./
 CMD     T Privs     Owner    Group        Size Date       Time     FILEID
         F rwxrwxrwx ********* *********     594 1996/04/24 12:43:53 .profile
         F rw------- ********* *********     105 1996/04/25 22:15:46 .sh_history
         F rwxrwxrwx rodnash   *********     408 1996/04/15 21:49:05 asg33
         d rwxrwxrwx rodnash   *********       0 1996/04/22 22:10:33 backup
         ? rwxrwxrwx rodnash   *********       0 1996/04/29 17:54:27 casg33
         F rwxrwxrwx rodnash   *********     250 1996/04/12 18:35:34 count
         ? rwxrwxrwx rodnash   *********       0 1996/04/29 18:41:16 fd
         F rwxrwxrwx rodnash   *********      12 1996/04/11 18:00:53 fred
         F rwxrwxrwx rodnash   *********     578 1996/04/24 13:43:17 hellorr
         F rw-rw-rw- rodnash   *********     648 1996/04/24 13:43:08 hellorr.c
         F rw-rw-rw- rodnash   *********     840 1996/04/24 13:43:16 hellorr.o
         F rw-rw-rw- rodnash   *********       0 1996/04/24 21:49:08 lastcommand
         d rwxrwxrwx dceadmin  *********       0 1996/04/02 22:22:56 level1
         F rw-rw-rw- rodnash   *********     019 1996/04/25 19:11:29 list
         F rw-rw-rw- rodnash   *********       0 1996/04/10 20:52:07 ls
         F rwxrwxrwx maintmo   *********     122 1996/04/10 20:17:46 mik
         F rw-rw-rw- rodnash   *********     263 1996/04/04 23:14:37 results
1= Help       2=          3= Quit   4= Delete      5=             6= Sort(DATE)
7= Backward   8= Forward  9=       10=Dirlist      11= XEDIT/List 12= Cursor
```

Figure 41. BFSLIST Sample Session

2.14.2 DU

When you list the contents of a directory with the OPENVM LIST command or the `ls` shell command, the file sizes are listed individually and there is no total, or tally. The DU tool lists the sizes of the files in a given directory in 512-byte-block increments. DU can be entered as is, with no arguments, to list the files and total blocks in the current directory. DU can also list the contents of another directory if an existing absolute path name is given as an argument.

In Figure 42 on page 70, the OPENVM LIST command shows the contents of the root directory followed by the output from using DU for the same directory.

```
OPENVM MOUNT /../VMBFS:VMSYS:ROOT/ /
Ready; T=0.01/0.01 14:58:32

OPENVM LIST /
Directory = '/home/maintmo'
Update-Dt  Update-Tm Type  Links          Bytes Path name component
09/18/1995 17:11:47   F        1            787 '.h'
04/10/1996 16:51:59   F        1            407 '.profile'
09/14/1995 14:15:31   F        1             11 '.rgy_editrc'
04/11/1996 16:58:08   F        1           1924 '.sh_history'
10/31/1995 14:43:29   D        -              - 'binop'
04/29/1996 16:49:08   F        1            250 'count'
04/29/1996 17:08:59   F        1            408 'fs'
10/10/1995 17:10:00   D        -              - 'greet2'
10/19/1995 13:35:14   D        -              - 'nes'
09/14/1995 13:29:07   D        -              - 'prime'
09/21/1995 09:37:59   D        -              - 'string_tree'
09/14/1995 13:22:00   D        -              - 'teldir'
04/29/1996 16:49:25   F        1            433 'testfile'
Ready; T=0.02/0.02 15:09:33
DU

 2      ./ .h
 1      ./ .profile
 1      ./ .rgy_editrc
 2      ./ .sh_history
 0      ./ binop
 1      ./ count
 1      ./ fs
 0      ./ greet2
 0      ./ nes
 0      ./ prime
 0      ./ string_tree
 0      ./ teldir
 1      ./ testfile

9   BLOCK TOTAL (512 Bytes per Block)
Ready; T=0.03/0.05 15:10:37
```

Figure 42. DU Output

The DU program has the following parts:

- DU EXEC
- DU REXX

You can use the DU tool in the shell, too. If you use a shell script to access the current directory or parse the one passed as an argument, the REXX EXEC may be called with the cms shell command.

Figure 43 contains the source of the DU shell script. As you can see, it only requires a single line of code.

```
#
# DU REXX Version of UNIX Disk Usage
#
cms du $1
```

Figure 43. DU Shell Script

Chapter 3. OpenEdition Planning Summary

Any CMS user can run a POSIX application, but to assure proper access to resources and to allow the management of POSIX data, the system administrator should first set up the OpenEdition for VM/ESA facilities.

The setup can be broken down as a series of installation or administration tasks, such as:

- File pool planning and installation, which includes installing the Shell and Utilities Feature
- System administration, which covers backups, system profiles, shared segments, and other global concerns
- User administration, which covers adding users, and setting up the POSIX database
- Security, which covers restricting privileges and providing proper security audits

These tasks can be further broken down into a series of planning and implementation topics. We have dedicated a chapter in this publication to each of these tasks.

3.1.1 Finding Additional Information

The following sections give a short overview of the topics discussed throughout the rest of this publication, and are the major areas of OpenEdition for VM/ESA planning and implementation:

- How to get the most out of the Byte File System can be found in Chapter 4, “Byte File System Installation” on page 75.
- Tasks involved with initializing and maintaining OpenEdition users can be found in Chapter 6, “User Administration” on page 137.
- Information on keeping OpenEdition for VM/ESA running can be found in Chapter 5, “System Administration” on page 107.
- How to keep the BFS secure can be found in Chapter 7, “Security” on page 159.

We recommend the following sources of information for additional reading:

- *CMS File Pool Planning, Administration, and Operation*, SC24-5751 should be considered a major source of information regarding your POSIX installation.
- In the *VM/ESA Installation Guide*, SC24-5748, there is a small step that is easily missed. It is at the end of the “Generate the File Pools” section and instructs you to run the BFSROOT EXEC.
- The *Program Directory for OpenEdition Shell and Utilities Feature for VM/ESA*, shipped with your shell and utilities tape, is another source of important installation information.

The following sections outline the basic planning steps required to get the most out of VM/ESA′s POSIX function.

3.1.2 Creating Byte File System File Spaces

At the center of OpenEdition for VM/ESA is the Byte File System (BFS). It provides the file system interface and semantics required by POSIX. BFS data is managed by a file pool server. A file pool server can manage SFS and BFS data simultaneously.

In general, BFS generation involves allocating storage space to one or more file spaces that will contain BFS data. The standard topmost file tree created resides in a system-provided file pool, VMSYS. Once the file system has been set up, you can add the FSROOT option to the POSIXINFO statement of an individual user′s CP directory entries. FSROOT specifies the file system (BFS file space) that will be mounted (made available as the root of the user′s directory tree) by CMS when the user accesses OpenEdition services.

The installation of the BFS allows for a flexible approach. A single BFS directory tree may be spread out over many file pool servers and file spaces.

3.1.3 Installing the Shell and Utilities Feature

The OpenEdition Shell and Utilities Feature for VM/ESA provides a UNIX-style command interpreter that conforms to the POSIX.2 standard. This feature is installed on top of the directories created during the installation of the BFS.

The installation uses VMSES/E and requires the following resources:

- A BFS file tree with enough space to hold the shell objects
- The POSIX user and group database that are initialized
- The LE/370 runtime library added to the global library list
- The LE/370 libraries at the latest service level
- The install user ID, 2VMVMZ10, prepared

3.1.4 General Preparation

Some of the following steps will make a big difference in the user friendliness of your system.

- Make sure the most current C libraries are located on a common disk, such as the MAINT 19E.
- Add statements to the users′ PROFILE EXECs or the system′s SYSPROF EXEC to mount the required BFS file spaces and to optionally enter the shell automatically.
- Set up a shell profile to define variables to tailor the shell environment for your system.
- Ensure that the proper backup system is in place for the BFS.
- Review all outstanding service to ensure a proper running system.

3.1.5 Assigning POSIX User IDs to VM Users

This step involves assigning a numeric value, the UID, to each user that will be using OpenEdition applications. You should make each user′s UID unique unless you have a specific reason to do otherwise. Users with the same UID will have the same access to files in BFS. The only UID with special meaning is the UID of zero (0), which denotes a superuser. Even here, a UID of 0 should only be assigned to the user ROOT.

File owners are chosen based on the first available match, alphabetically speaking, in the CP directory. If a user named BELMA was added to the CP directory and given a UID of 0, all of the files owned by ROOT would now appear owned by BELMA. BELMA is what would be returned from the `ls` command. If user ALBERT is also assigned a UID 0, the ownership would appear to be ALBERT since this is the first user with a UID of 0 CP sees when it scans the user database.

If an ESM is installed to maintain the POSIX user database, see the ESM instructions for assigning UIDs. Otherwise, the UIDs are assigned in the user entries in the CP directory using the POSIXINFO statement. Any user not explicitly assigned a UID will automatically be assigned a default UID of 4294967295 (X'FFFFFFFF').

3.1.6 Defining POSIX User Groups

This task involves assigning a group ID (GID) to a group name. If an ESM is not installed to handle the POSIX group database, GIDs are assigned in the global definition section of the CP directory with a POSIXGROUP statement. Duplicate group names are not permitted, but multiple groups may have the same GID. Groups that have the same GID are considered to be the same when performing file access permission checking; however, they are treated as different groups during database queries such as `getgrnam()` and `getgrgid()`.

Certain queries that require a GID as input may return information about a group other than the intended one if identical GIDs are used. Use care when assigning the same GID to more than one group.

3.1.7 Assigning VM Users to POSIX User Groups

The assignment of users to user groups should be based on common access to data. To share BFS files with other users, give selected file access permissions to the members of the file's user group. User groups define the set of users that have common file use needs.

A user can be assigned membership in multiple groups by specifying multiple group names or GIDs on the POSIXGLIST statement. To avoid ambiguity if multiple groups are defined with the same GID, it is recommended that groups be specified by group name rather than by GID.

Some commands return the name of the user group. These commands include the OPENVM LISTFILE command provided by CMS and the `ls` command provided by the OpenEdition Shell and Utilities Feature for VM/ESA. The group names are defined in the global definitions section of the CP directory using the POSIXGROUP statement.

3.1.8 Selecting Additional Security Features

Additional controls are provided that limit the use of some POSIX features on a system-wide or per-user basis. There is a system configuration file statement that defines attributes and permissions for all users on the system. The QUERYDB specification defines whether the users are ALLOWed or DISALLOWed to query other user's POSIX database information. The EXEC_SETIDS specification defines whether users are ALLOWed or DISALLOWed to have their POSIX IDs changed by a POSIX exec() function call or set-ID file.

The security concern with allowing a user to execute set-ID files is that the user acquires the authority associated with the file. A malicious user could interrupt a program and access or change data that this virtual machine would not normally have available.

There are several implications that should be noted if you choose to use these additional controls:

- Any user who is not allowed to execute set-ID programs will not be able to use the `mailx` utility provided with the OpenEdition Shell and Utilities Feature for VM/ESA. This utility allows users to exchange notes in a UNIX-like fashion.
- Any user who is not allowed to query the user database information for other users will not be able to use all of the options of the `ls` utility, which is also provided with the OpenEdition shell, to find user and group name information related to files.

Chapter 4. Byte File System Installation

Most of the installation considerations regarding OpenEdition for VM/ESA pertain to the BFS. Before starting the installation of the BFS, there are important points to consider that could affect your file pool installation. This chapter discusses the advantages and disadvantages of different BFS installation scenarios, and the importance of mount points.

The main differences between the installation scenarios derive from the setup of the BFS server. The following installation scenarios to set up your BFS are discussed:

- Using the default file pool servers for BFS data
- Using a customized BFS server
- Combining multiple BFS servers

Of course, your flexibility does not stop here. The implementation of the BFS can be taken well beyond the complexity described in this chapter. The scenarios described are intended to point to the most general of the BFS exploitation possibilities and to start you off in creating the best possible implementation for your environment.

No matter what scenario you use, keep in mind that the major tasks pertaining to the BFS are:

- Preparing the file pool server machines
- Creating the base directory trees using BFSROOT
- Installing the Shell and Utilities feature
- Creating /home directories, profiles, and links

Before discussing the actual installation steps, an overview of some basic file pool concepts is given.

Readers familiar with the Shared File System can skip the next section.

4.1 File Pool Overview

The POSIX.1 specification defines a hierarchical file system. In OpenEdition for VM/ESA, this implementation is called the Byte File System (BFS). Data for the BFS is stored in the same repository as the Shared File System (SFS).

The file pool server maintains data in a file pool, which is made up of one or more storage groups. At the file pool and storage group level, BFS and SFS data can coexist.

File pool data consists of file spaces and each file space can belong to individual users. There cannot be a mix of BFS and SFS data at the file space level. That is, a user's file space must contain either byte data or SFS data.

Users may use their own SFS file spaces while using and owning files in other BFS file spaces. A user may have multiple file spaces existing in several file pool servers. In the POSIX world, a single file space may be an entire file system, or simply one section of a larger tree.

As of VM/ESA Version 2 Release 1.0, file pool servers have four functions, namely:

- SFS support
- BFS support
- CRR
- FIFO support

FIFO and BFS support are new for VM/ESA Version 2. A FIFO server is a special server dedicated for FIFO BFS files. It can be used in situations requiring better FIFO performance.

4.1.1 File Pool Structure

A file pool provides centrally managed disk storage that many users can share. The file pool has several minidisks, each with specific functions, that are defined as follows:

- Control Minidisk

  This minidisk contains a map of all the blocks of user data (file pool control data), that is stored in the file pool. It is created when the file pool is generated at install time.

- Catalog Storage

  This catalog consists of one or more minidisks that contain information about the data that exists in the file pool, such as who the owner is and who is authorized to access what resource. Catalogs reside inside the file pool control data.

- Log Minidisk

  This disk contains the file pool log data. Two minidisks maintain a record of the directory changes made in the file pool.

  The concept of a work unit is central to the integrity of the data stored in a file pool. A work unit is a group of related operations that can be committed or rolled back as a unit. Multiple work units can be active in a single client machine. The work unit is reusable, and new operations may be associated with the same work unit, so all operations associated with this work unit are committed or rolled back as a group. Operations on a BFS are seen as a single work unit.

- User Storage Groups

  These groups contain all the user data. They are part of the file pool repository data. User data may grow to cover many server minidisks. To maintain control over which users have files on which DASD volume, the server associates disks by storage group.

Figure 44 on page 77 shows the logical separation of the different minidisks within a file pool.

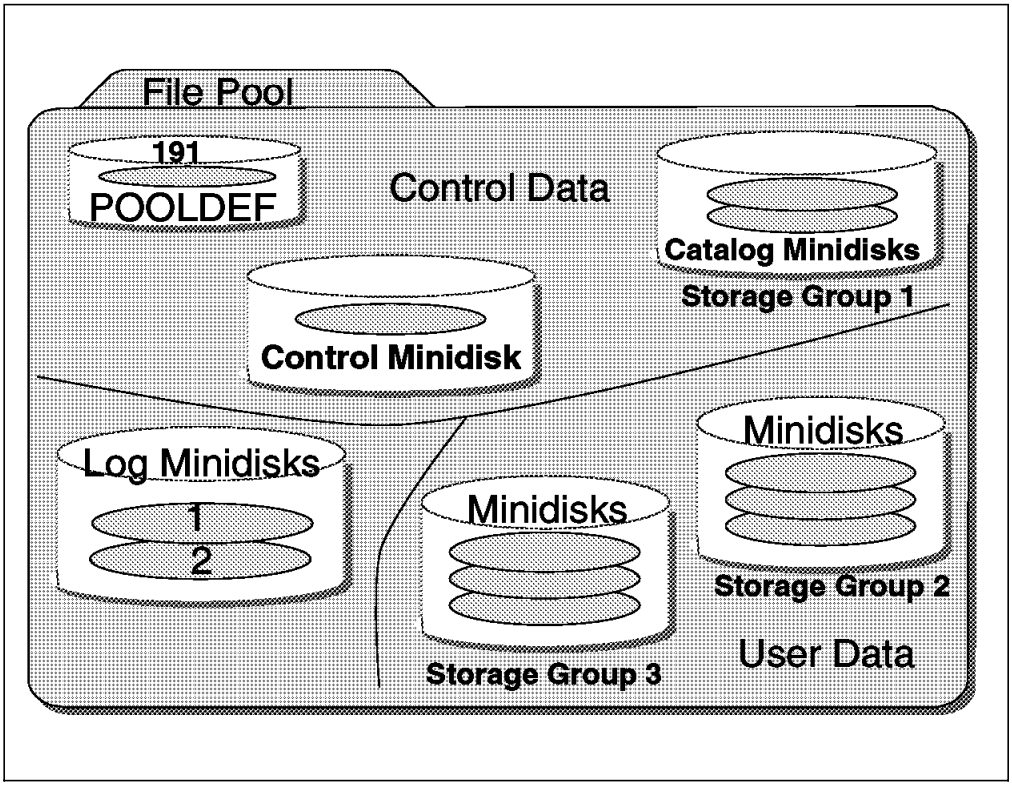


Figure 44. File Pool Structure

In each file space, the user sees the data arranged by directory. The differences between conventional record file data and byte file data begin at this point. Though both systems offer a directory view, the directories and data are controlled in completely different ways.

4.1.2 The File Pool Administration Machine

The file pool administration machine is a user that is granted file pool administration authority. From this machine, you can control the resources in the file pool and execute all the maintenance and administrative functions. Backup of the file pool, creation of BFS spaces, user enrollment, and user space allocation are some of the activities managed from the administration machine. Administration can be performed when the file pool server is running in multiple user mode.

Due to the different types of operations involved in the administration of a file pool, you typically must implement various administrative users to control the operation of a single file pool. One administrator may handle users′ needs, another may handle system backup requirements. The administration functions could also be handled by another virtual machine, such as with the DFSMS/VM product.

The file pool administrator bypasses all POSIX authorization checking applied to objects in a BFS file space.

The major difference between superuser and a file pool administrator is the scope. Superuser privileges cover all file pools containing BFS file spaces, but do not extend to SFS data that may be stored on the same servers. The administrator has access to all data in a given file pool.

A complete list of the administrative tasks is given in the *CMS File Pool Planning, Administration, and Operation*, SC24-5751.

4.2 File Server Planning

During the installation of VM/ESA, you are required to decide which of the default servers and file pools you want to install. In using the default servers, little extra configuration is required. However, your decision making should take in consideration your system's future role. A VM system consisting of thousands of users will require more than default servers.

To plan the installation of a file pool used for BFS data, remember the following points:

- If you choose to install the IBM-default file pools and servers, it is a good idea to install all three (system data, user data, and recovery).
- If you choose to install your own set of file pool servers, you must consider the additional factors of how much storage is needed, how many users will be served, and how to customize the install configuration files to get the results you want.

Depending on the needs of your users, SFS and BFS data can be separated at the file pool level, or restricted to a storage group within a file pool server handling SFS and BFS data. It is not recommended to mix the data beyond this level (although still possible to do so). Implementation of the BFS allows several users to build directories and manipulate data within a single file space. In fact, a single file space (although this is not recommended) can contain an entire BFS file system.

The following steps are recommended for a well planned and successful BFS implementation:

- Install the BFS directories using the default file pool servers.
- After you have become familiar with the BFS and OPENVM commands, install a new file pool that will contain only BFS data. This machine will hold the user's `/home` directories.
- Using external links and extra mount commands, connect the new user-data BFS to the default servers to provide a seamless file system name space.

Hints and Tips

When you are new to the BFS, it is easy to play with commands to the point where important data is lost. Do not make this mistake. Gather experience with the BFS before putting it into production, or create a sandbox server for experimentation.

4.3 Default Byte File System File Pool Servers

In a standard VM/ESA installation, there are three file pool server machines: VMSERVS, VMSERVU, and VMSERVR. These server machines do the following:

VMSERVS The VMSERVS machine controls the resources for the VMSYS file pool that contains the system files.

VMSERVU The VMSERVU machine controls the resources for VMSYSU, (that is, the file pool for users).

VMSERVR The VMSERVR machine controls the resources for the VMSYSR file pool, which is also known as the Coordinated Resource Recovery (CRR) server.

For small systems, the default file pool servers are the most efficient in DASD utilization. All of the system's shared file needs can be compressed into two file servers and one recovery machine. The default machine prevents any file sharing exposures between multiple VM systems since the servers are unable to be started as a global system resource.

Following are some considerations regarding the storage of BFS data in the VMSYS and VMSYSU file pools.

4.3.1 VMSYS File Pool Server

This file pool is managed by the VMSERVS virtual machine and is the default repository for system owned data. Regarding the BFS, it will contain static data, such as executables.

If you are planning to use Data Facility System Managed Storage/Virtual Machine (DFSMS/VM), you must install VMSYS.

The VMSYS file pool is also the default repository of data from various components of VM/ESA itself, such as GCS and TSAF.

Root Byte File System File Space in VMSYS

The default installation process establishes the BFS root file space and root directory in the VMSYS file pool. This file space is called ROOT, and a CP directory entry for a user ID of ROOT containing UID and GID essentials is required. The DIRPOSIX utility, described in section "DIRPOSIX Utility Program" on page 141, can be used to add the ROOT user ID to an existing directory.

To access the entire BFS file tree, the root directory must be mounted. This mount establishes the BFS root and can be made with an FSROOT directory statement, or the OPENVM MOUNT command. In either case, the default VMSYS root will be:

```
/../VMBFS:VMSYS:ROOT/
```

Keep in mind that there are two levels of BFS installation:

- Defining the root directories with the BFSROOT EXEC
- Installation of the shell and customization of the environment for production use

During the VM/ESA installation process, the BFSROOT EXEC is used to create the root file space and directory in the VMSYS file pool. It also creates two other file spaces in the VMSYSU file pool (TMP and VAR), and these additional file spaces are joined with mount external links.

When the default root and basic file directories are in place, the OpenEdition Shell and Utilities Feature for VM/ESA installation process can proceed. The installation creates the `/bin` directory as well as other BFS objects.

Byte File System Root File Space Not in VMSYS

If you need to override the default file systems, you have to build a modified version of the BFS LOADBFS configuration table located on the MAINT 193 minidisk. This control file resolves where the BFS root structure is loaded by the BFSROOT EXEC. The modification of the control file will replace the VMSYS and VMSYSU file pool names with the ones that you want to create.

If you move the BFS root file system to another structure, or select multiple roots on a single VM system, then install or provide connections to the OpenEdition Shell and Utilities for each unique file tree. It is important that your end-user's environment appears consistent from server to server, file system to file system.

4.3.2 VMSYSU File Pool Server

This file pool is managed by the VMSERVU virtual machine and is the default repository for user data. Regarding the BFS, it will contain the dynamic file trees. These directories typically include logs, user data on small systems, and other BFS objects that are often updated.

In addition to the default users enrolled in the installation process, two extra file spaces (TMP and VAR) are created on the file pool by BFSROOT. They are described as follows:

TMP The TMP file space is used to allocate space to the `/tmp` BFS directory. This path is located with a mount external link (MEL) to the VMSYS file pool. After mounting the root directory, any application moving into the `/tmp` directory communicates automatically to the VMSYSU file pool through a transparent mount.

VAR Similar to TMP, the VAR file space is used for dynamic storage by various BFS programs. It also has a unique file space and is located by a MEL.

The ultimate location for these two file spaces is defined in the BFS LOADBFS control file used by the BFSROOT EXEC.

Since both VMSYS and VMSYSU are used for the BFS file tree, the system availability of their service machines is critical.

4.3.3 VMSYSR File Pool Server

This file pool is managed by the VMSERVR virtual machine, and it is used for Coordinated Resource Recovery (CRR). The BFS does not use the services of CRR, so it is not dependent on the installation of VMSYSR. However, the absence of the VMSYSR or the CRR recovery routines affects the performance of other operations performed on the BFS, such as backup. Therefore, install VMSYSR to avoid any performance degradation. VMSYS and VMSYSU may continue to house CMS SFS data.

4.4 A New File Pool for BFS Data

Creating a separate file pool dedicated exclusively for BFS data is recommended. It can be a natural extension of the default VMSYS and VMSYSU file pools used to house the `/home` directories, or it can contain all the BFS directories from `root` on down the file tree.

The file pool server installation definitions have similar characteristics to those of the IBM-default VMSYSU file pool and VMSERVU server. The steps necessary to define a single BFS server are described in the following sections of this chapter and in *CMS File Pool Planning, Administration, and Operation* SC24-5751.

The reasons behind defining a single shared file server for exclusive dedication to BFS data is summarized in section 4.4.7, “Advantages and Disadvantages of Using a Dedicated BFS Server” on page 90.

4.4.1 Definitions Used for a File Pool Server

The following list is a brief description of the main steps and definitions required to generate a new file pool and use it as a repository for BFS data.

- Estimate the maximum number of users.

  This is necessary to generate the file pool, but it does not limit the number of users or BFS file spaces that can enroll in the file pool. This value is used on the MAXUSERS statement in the *server* POOLDEF file used by FILESERV GENERATE. The value must be no less than 5 and no greater than 32767.

- Estimate the maximum number of minidisks.

  Because each minidisk added will probably be large enough to support more than one user, the value of MAXDISK should be equal to or a little smaller than the number of MAXUSERS.

  Never use too small a value because you will need to regenerate your entire file pool in order to increase it. For most file pools, a value of 500 is enough.

- Determine the directory MAXCONN value.

  A user virtual machine communicates with the BFS server virtual machine over Advanced Program-to-Program Communication (APPC) connections. Two APPC paths are used by each POSIX process during its execution, and each mounted file system also requires two paths. This means that if the user′s directory tree is composed of elements managed by many file pools, the user should have a MAXCONN value in the user′s CP directory of at least 64 (and perhaps more).

- Determine the initial DASD allocations.

  When defining a new server, estimate the initial minidisk allocation for the server and its structure. Many disks can be allocated to satisfy the space requirements of your BFS data. One of the values that limits the size of the file pool is the size of the control minidisk. This control minidisk holds the file pool control data, which is a map of the use of each 4KB (4096 bytes) block of DASD space. If the control minidisk becomes full, the machine will terminate processing.

  1. Determine control minidisk size.

     The size of the control minidisk determines how many bytes the file pool will hold. Try to estimate the size of the minidisk you will need. Too high an estimate results in unused minidisk space. Too low an estimate means that you will have to regenerate the entire file pool if the minidisk becomes full.

     To determine the size of the control minidisk, determine how many bytes (characters) the file pool must hold. This number will vary based on the number of users and the number and size of the files they create.

Estimate the size of your file pool in gigabytes (GB). A gigabyte is 1,073,741,824 bytes (about a billion). If you group those bytes in blocks of 4KB you will have 262,144 blocks per gigabyte. Table 8 on page 82 contains an estimate of the size of your file pool and the size of the minidisks needed to support it. This table is calculated based on MAXUSERS=1000 and MAXDISK=500.

Table 8. Estimates of Control Minidisk Sizes

Size of File Pool (gigabytes)	Size of File Pool (4KB Blocks)	Size of Control Minidisk (512-byte blocks)
0.1	26215	1821
0.5	131072	1905
0.7	183501	1943
1.0	262144	2008
5.0	1,310,720	2831
10	2,621,440	3862
50	13,107,200	3862
100	26,214,400	22407
1000	262,144,000	207865

If MAXDISK is greater than 500, use the following formula to add to the number in Table 8.

$$((\text{MAXDISK} - 500) \div 100) \times 4$$

For a repository file pool server, the minimum size of the control minidisk is 5 cylinders on any count-key-data (CKD) DASD or 5000 blocks on any Fixed-Block (FB) device.

If MAXUSERS is greater than 1000, you can use the following formula to add to the number in Table 8.

$$((\text{MAXUSERS} - 500) \div 100) \times 4$$

For more complete information about the number of byte blocks per cylinder based on the device type, refer to *VM/ESA Planning and Administration*, SC24-5750.

The placement of the control minidisk is not critical, but it is a good idea to place it on the volume that contains storage group 1. The control minidisk and storage group 1 are recovered together, so if there is a DASD error on a volume, they can be reloaded synchronized.

2. Determine the log minidisk allocation.

   Every file pool must have two identically-sized file pool repository log minidisks. To avoid problems related to difference in size or in device type, the definition of these two minidisks should be done on the *same device type*, and on *separate physical devices*. The minidisk should be large enough to hold the log data and the file pool backup control data. The backup control data is used to restore the control data in the case of a media error. The following simple formula can be used to estimate the size of this minidisk:

$$\text{LOGSIZE} = \text{HOURS} \times \text{ACTIVE\_USERS} \times 0.08$$

This formula estimates the LOGSIZE, based on the number of HOURS the file pool is available and the number of the users that are working on the file pool. The value of 0.08 is an estimate of the rate at which the file pool log repository log data is generated.

3. Determine the file pool repository size.

   The file pool repository must be large enough to hold all the changes to the control data when a backup is taken.

4. Determine the work disk allocation.

   With many VM servers, the server machine requires a work disk. In this case, the disk will hold the POOLDEF file, which contains the file pool definitions, and the DMSPARMS file, which contains startup parameters of the file pool. These files will be created in the file pool generation process.

   You can also define this work disk in another Shared File System, depending on your particular installation requirements.

   For most installations, five cylinders is enough.

5. Determine the catalog minidisk allocation.

   The storage group 1 minidisk is a small, I/O intensive area. It is best to spread storage group 1 over only enough volumes to avoid an I/O bottleneck. A rule of thumb is to spread storage group 1 across USERS ÷ 130 DASD volumes (the USERS value is specified in the startup parameters). There is no limit to the number of minidisks that can be added to satisfy the storage requirements of this object.

   As a minimum size, approximately 500 bytes are required for each BFS file. A size of 5 cylinders is enough for a small server.

6. Determine the user minidisk allocation.

   You can add as many minidisks as you need to the file pool to satisfy the users′ BFS storage needs. Check the actual utilization of your file pool with the QUERY FILEPOOL STORGRP command. Remember that file space is not synonymous with *user*, as it sometimes is with SFS data. Many larger VM installations decide to give each BFS user ID a unique file space for its private data.

   Note that the more storage groups you have per volume of data, the less time it takes to recover one of them, since they are smaller. For optimum I/O load balancing, avoid disks with high usage, make sure that all volumes of a storage group are of the same device type, and give the storage group about the same amount of space on each volume.

   At generation, at least one disk is required. It is easy to add a minidisk if needed at a later time by using the FILEPOOL MINIDISK or FILESERV MINIDISK command.

4.4.2 Define the Server Machine

Once you have all the information required to create a file pool, the next step is building it. Using your local procedures to update the VM/ESA CP directory, you must define a file pool server and give it the characteristics needed to perform as a server. Among these specifications are several that need special attention. They are referred to in Figure 45 on page 84.

Note that the generation of a BFS server and SFS server are very similar.

```
USER VMSERBFS VMSERBFS 32M 32M BG                                    1
   ACCOUNT 1 VMSERBFS
   IPL CMS
   IUCV ALLOW
   IUCV *IDENT RESANY GLOBAL
   MACH XC
   OPTION MAXCONN 2000 NOMDCFS APPLMON ACCT QUICKDSP SVMSTAT         2
   POSIXOPT SETIDS ALLOW                                             3
   SHARE REL 1500                                                    4
   XCONFIG ADDRSPACE MAXNUMBER 100 TOTSIZE 8192G SHARE               5
   XCONFIG ACCESSLIST ALSIZE 1022
   CONSOLE 0009 3215 T MAINTMO                                       6
   SPOOL 000C 2540 READER *
   SPOOL 000D 2540 PUNCH A
   SPOOL 000E 1403 A
   LINK MAINT 0190 0190 RR
   LINK MAINT 0193 0193 RR
   LINK MAINT 019D 019D RR
   MDISK 0191 3390 582 3 210RES MR XXXXXXX XXXXXXX
   MDISK 0301 3390  585 009 210RES WR XXXXXXX XXXXXXXX              7
   MINIOPT NOMDC
   MDISK 0302 3390  594 014 210RES WR XXXXX XXXXX
   MINIOPT NOMDC
   MDISK 0303 3390  609 014 210RES WR XXXXX XXXXX
   MINIOPT NOMDC
   MDISK 0304 3390  624 003 210RES WR XXXXXXX XXXXXXXX
   MDISK 0305 3390  627 006 210RES WR XXXXX XXXXX
   MDISK 0306 3390  633 025 210RES WR XXXXX XXXXX
   MDISK 0307 3390  552 030 210RES WR XXXXX XXXXX
   MDISK 0308 3390 1567 050 210RES WR XXXXX XXXXX
```

Figure 45. BFS Server Direct Example

1 32MB is the recommended storage size needed to run the server, but it will run on less. CP class B is required to attach tapes when the backup task is performed directly from the server machine.

2 OPTION MAXCONN determines the number of concurrent IUCV connections allowed to the server. This value could be calculated using the following formula.

$$MAXCONN = (BFS\_USERS \times 12) + DISKS$$

BFS_USERS should be the number of concurrent BFS USERS that may be expected during peak activity. DISKS is the total number of minidisks allocated to the file pool. The value of 2000 should be enough for medium file pools but not for very large file pools (which is defined as more than 700 users accessing a single file pool).

The NOMDCFS operand allows the server to use minidisk caching at a rate that is not limited by the fair share limit.

APPLMON operand allows the server to generate information for performance monitoring. The server processing enters a DIAGNOSE X'DC', so the use of the APPLMON is required.

The ACCT option is required if you will use the accounting facilities.

The QUICKDSP option causes the server to be added to the dispatch list immediately when it has work to do.

The SVMSTAT option specifies that the server is a service virtual machine, and SVMSTAT allows the server's monitor statistics to be reported separately from the other users' machines.

3 POSIXOPT SETIDS ALLOW allows the server to execute code that uses the set-ID functions.

4 The statement SHARE REL 1500 places the server in a preferred position in the dispatch queue. This statement is common for server machines. At times, it is overused.

5 The XCONFIG ADDRSPACE statement authorizes the server to create data spaces and sets limits on the number and sizes of those data spaces. As a rule of thumb, define the maximum number of data spaces you want as five times the number of the directories you plan to make eligible for data spaces.

The XCONFIG ACCESSLIST ALSIZE 1022 statement sets up space for the server's host access list. This ACCESSLIST statement, as well the ADDRSPACE statement, should be defined if you intend to use data spaces in the file pool. Since BFS directories cannot be set as dircontrol, a file pool server dedicated for a BFS will not need these statements, but including them helps to keep the installation as generic as possible.

6 The CONSOLE statement defines a secondary user ID. This is needed, because most of the time this server runs in a disconnect state. The secondary user support is known as Single Console Image Facility (SCIF).

7 All the MDISK control statements for the file pool include both a read and write password, and have an access mode of WR. Passwords are only needed when an ESM is not active. The MINIOPT and NOMDC control statements inhibit expanded storage caching.

Hints and Tips

Do not add LINK control statements to the VM/ESA directory for the file pool minidisk. File pool server machines automatically link the disks.

4.4.3 Define the Administration Machine

This is the machine that issues the administrative commands to the server. Use one of your existing administrator virtual machines for this. (It is a good idea to use the same one that receive the messages and that is the secondary user.)

An administrator has access to all data residing on a particular server, and it is able to override any file permission settings. The administrator is, in effect, a superuser to that file pool server.

Figure 46 on page 86 shows the administration machine of the VMSERBFS server, and the VMBFS file pool with its associated minidisks.

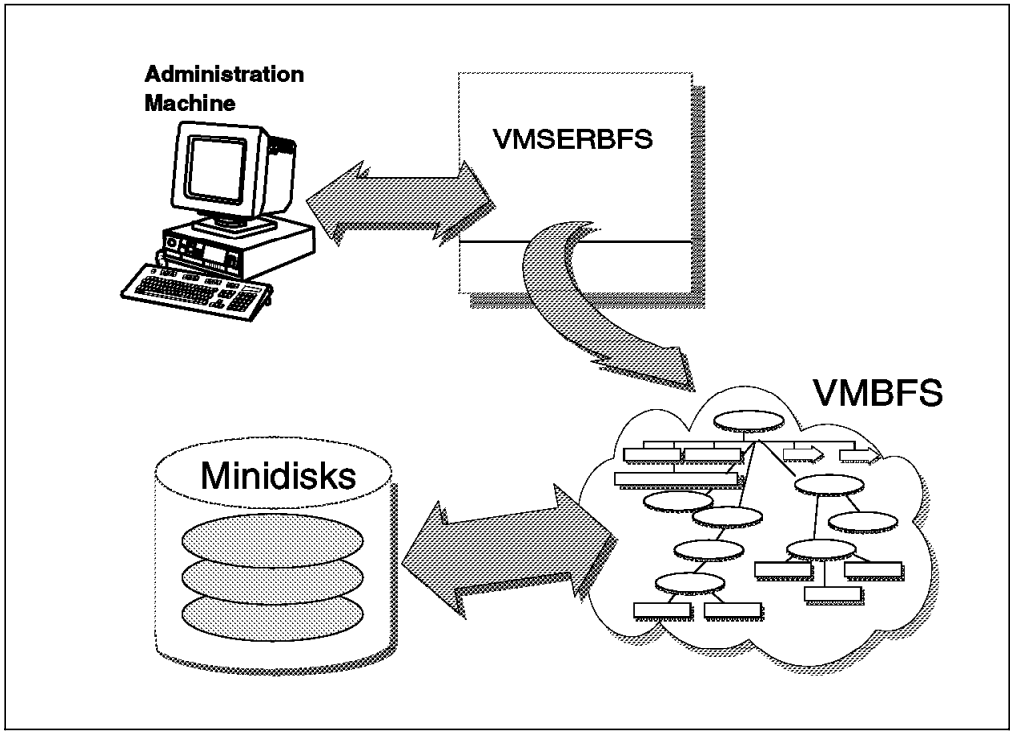


Figure 46. The VMSERBFS File Pool Server Supporting the VMBFS Name Space

4.4.4 Generation of a File Pool

Once you have defined the server machine, define the startup parameters to be used during the file pool generation. This is required by the FILESERV command. Any time you use the FILESERV command, it will use these startup parameters. They are held in a file called *servername* DMSPARMS. The server name is the name of the CP user ID, not the name of the resource. The required definition of the startup parameters are in Figure 47.

```
ADMIN MAINTMO          1
BACKUP                 2
SAVESEGID CMSFILES     3
FILEPOOLID VMBFS       4
USERS 100              5
NODFSMS                6
```

Figure 47. Server DMSPARMS Example

1 The ADMIN statement identifies the user IDs that receive administrator authority from this file pool server.

2 The BACKUP startup parameter indicates that the backup is being handled by the file pool repository facilities, and that automatic backup will occur when repository log usage reaches its threshold. If NOBACKUP is in effect, you cannot back up the control data.

3 The SAVESEGID indicates that the server uses code that has been loaded into a physical saved segment. The default physical saved segment for the server code is CMSFILES.

4 This startup parameter defines the file pool name. The rule to name a file pool is the same as that for naming files: you may use up to eight alphabetic or numeric characters without imbedded blanks. Pick names easy for users to remember, and follow a logical convention.

5 The USERS statement is the maximum number of users expected to be connected to the file pool during the same instance.

6 Use the NODFSMS statement to indicate that the server will not use DFSMS to manage and administer the file pool. DFSMS can manage BFS file data.

After you create the startup parameters file, you can generate the file pool using the following command:

```
FILESERV GENERATE
```

FILESERV will then format and reserve the minidisks used by the file pool. FILESERV will prompt you for the device addresses of the minidisk, and then store them in a file called *resourceid* POOLDEF, as shown in Figure 48.

```
MAXUSERS=1000                                                          1
MAXDISKS=500                                                           2
DDNAME=CONTROL          VDEV=301                                       3
DDNAME=LOG1             VDEV=302                                       4
DDNAME=LOG2             VDEV=303                                       5
DDNAME=BACKUP   DISK    FN=CONTROL     FT=BACKUP    FM=A               6
DDNAME=MDK00001         VDEV=304       GROUP=1   BLOCKS=532           7
DDNAME=MDK00002         VDEV=305       GROUP=2   BLOCKS=1070          8
DDNAME=MDK00003         VDEV=306       GROUP=2   BLOCKS=4487
DDNAME=MDK00004         VDEV=308       GROUP=3   BLOCKS=8983
```

Figure 48. FILESERV POOLDEF Example

The first screen that you see contains the default values. You have to update these values to the ones you calculated for your file pool. Here you can see that all the pieces are coming together:

1 This is the value of MAXUSERS that you calculated before.

2 This is the value for MAXDISKS.

3 The VDEV address that you specify in the next DDNAME statements identifies the minidisks of the file pool server. Here, 301 is the CONTROL minidisk.

4 DDNAME=LOG1 defines minidisk 302 as the first log file minidisk.

5 DDNAME=LOG2 defines the second log file minidisk as residing on the 303 minidisk.

6 DDNAME=BACKUP tells the server that the backup of the control data will reside in a file called CONTROL BACKUP on the A-disk. (You can also define a tape unit as your backup destination, but this is not recommended because of lack of performance.)

7 The MDK00001 statement defines that the control data or storage group 1 will reside on the 304 minidisk. You could omit the BLOCKS parameter. The FILESERV command updates this field automatically when it formats this disk.

8 The remaining statements are used to insert the user's minidisks. As you see, they belong to different storage groups. This mixture of different locations provides a convenient recovery and backup. You have control of where is the data located.

If you are defining a file pool exclusively for BFS data, check your definitions against those described in *VM/ESA CMS File Pool Planning, Administration, and Operation*, SC24-5751.

When you have finished customizing the POOLDEF file, file the results and FILESERV will resume executing.

4.4.5 File Pool Server Profile EXEC

The PROFILE EXEC for this server machine is shown in Figure 49.

```
/* PROFILE EXEC FOR BYTE FILE SYSTEM SERVER */
address command
'SET AUTOREAD OFF'
'CP SET RUN ON'
'CP SPOOL CONSOLE START TO MAINTMO'
'SET PF1 RETRIEVE'
'SET PF2 IMMED FILSERV START'
'SET PF3 IMMED STOP NOBACKUP'
'SET PF4 IMMED #CP DISC'
'ACCESS 193 B'
if word(diagrc(24,-1),2) = 2 then 'EXEC FILESERV START'
```

Figure 49. Server PROFILE Example

The PROFILE EXEC allows the machine to start automatically after an IPL and completes the generation steps of a file pool server.

Set the server to work in MULTIPLE USER MODE to allow others access to it.

4.4.6 Create a Customized BFS

If you have installed VM/ESA Version 2 Release 1.0 from a starter system, there are some basic CP directory POSIX statements that are predefined and that will allow you to initialize the BFS. If you are migrating from a previous release of VM, you should run the DIRPOSIX utility. DIRPOSIX and information on the preparation of the CP directory can be found in section 6.3.1, "First Time Handling of the Directory Source File" on page 141.

Before using a BFS, there must be an initial set of directories and objects created. This is done using the BFSROOT EXEC. When you are using a unique file pool (non-default values chosen for the installation), the following steps are required:

1. Ensure that you have successfully generated the file pool server.
2. Enter LOGON BFSADMIN (or any user with file pool server administrator authority).
3. Enter LINK MAINT 193 193 RR
4. Enter ACCESS 193 T/T
5. Enter COPY BFS LOADBFS T BFS LOADBFS A
6. Enter XEDIT BFS LOADBFS A
7. Change the file pools in the ENROLL command to the BFS you intend to use. We decided to call our file pool VMBFS.

Figure 50 on page 89 shows the state *before* making the changes:

```
ENROLL ROOT                root     system -rwxrwxrwx ( VMSYS   2  1000
ENROLL TMP                 root     system -rwxrwxrwx ( VMSYSU  2  1000
ENROLL VAR                 root     system -rwxrwxrwx ( VMSYSU  2  1000
```

Figure 50. BFS LOADBFS File

The resulting changes (the *after* state) are shown in Figure 51.

```
ENROLL ROOT                root     system -rwxrwxrwx ( VMBFS   2  1000
ENROLL TMP                 root     system -rwxrwxrwx ( VMBFS   2  1000
ENROLL VAR                 root     system -rwxrwxrwx ( VMBFS   2  1000
```

Figure 51. BFS LOADBFS Modified File

8. Change the first two EXTLINK statements and replace the file pools with the names you are using.

   This is required because the /tmp and /var file trees are now in the same file pool as ROOT. The EXTLINK remains, since the TMP and VAR users are both enrolled in the server with separate BFS file spaces assigned to them. If these directories were part of the same file space, then an SLINK statement with no MOUNT option could be used when a fully qualified path name is supplied. It is recommended you use external links even when all the directories are contained within the same file pool server.

   Figure 52 shows the state *before* making the changes:

```
EXTLINK /tmp  MOUNT        root     system   .           ( /../VMBFS:VMSYSU:TMP
EXTLINK /var  MOUNT        root     system   .           ( /../VMBFS:VMSYSU:VAR
```

Figure 52. BFS LOADBFS Standard File

The resulting changes (the *after* state) are shown in Figure 53.

```
EXTLINK /tmp  MOUNT        root     system   .           ( /../VMBFS:VMBFS:TMP
EXTLINK /var  MOUNT        root     system   .           ( /../VMBFS:VMBFS:VAR
```

Figure 53. BFS LOADBFS Modified File

You may want to choose a more descriptive name than VMBFS for your BFS server.

Whether you use a modified BFS LOADBFS or not, you must now execute:

```
BFSROOT
```

to build the static and variable file trees for VMBFS.

Hints and Tips

BFSROOT EXEC

- Before installing the OpenEdition Shell and Utilities Feature for VM/ESA, you must run the BFSROOT EXEC, located on the MAINT 193 disk.
- However, for the default file pools (VMSYS/VMSYSU), this EXEC may have been previously run during VM/ESA installation. Check with the system programmer responsible for the VM/ESA installation; otherwise, you may see many *RC=28 Object already exists* warning messages during BFSROOT execution.
- Make sure the user ID that is running the BFSROOT EXEC is enrolled as SFS ADMIN in the file pool. (If not enrolled, a code of 0 will be returned, but not all definitions will be correct.)
- A UID is not required in this case, since the ADMIN authority contains the needed privileges.

Figure 54 shows the directories created by the BFSROOT EXEC. The OPENVM LIST command lets you look at the directories without the use of the shell and utilities.

```
OPENVM MOUNT /../VMBFS:VMSYS:ROOT/ /
Ready; T=0.01/0.02 17:09:21
OPENVM LIST /
Directory = '/'
Update-Dt  Update-Tm Type  Links          Bytes Path name component
04/15/1996 10:20:42   D        -              - 'dev'
04/15/1996 10:20:39   D        -              - 'etc'
04/16/1996 09:44:52   D        -              - 'home'
04/15/1996 10:20:41   L        -              - 'lib'
04/15/1996 10:20:39   D        -              - 'opt'
04/15/1996 10:20:40   E3       -              - 'tmp'
04/15/1996 10:20:41   L        -              - 'u'
04/15/1996 10:20:40   D        -              - 'usr'
04/15/1996 10:20:40   E3       -              - 'var'
Ready; T=0.01/0.02 17:10:37
```

Figure 54. BFS File Structure Example

4.4.7 Advantages and Disadvantages of Using a Dedicated BFS Server

The use of a dedicated file pool server for BFS data has both advantages and disadvantages, which are described in this section:

Advantages

Following are reasons to consider using a single separate file pool server for BFS data:

- The size of the file pool can be set up to match the desired scope of your BFS workload.
- Intrusive outages, such as DASD failures, can be isolated to only one critical business function.
- The administrator will only have access to one type of data. You can define precise POSIX and CMS administration boundaries.

- The backup and restoration of data is more flexible. If you need to restore a file pool storage group, it will contain only BFS data.
- BFS data runs under a unique locking structure that increases the number of I/Os during file read and write operations. Isolating BFS data will have a positive effect on normal SFS performance.
- The uniqueness of directories and objects is guaranteed. Therefore, you can find directories and objects quickly, knowing data and products are unmixed.
- If you plan to install Distributed Computing Environment (DCE) on your system, you may want to separate it from the regular file pools. This facilitates testing and insures a unique name space.
- Migration to new levels of code is easier. A single new server could be used for beta testing before being put into production.
- The default install places data in the two file pools that can never be used for remote access.

Disadvantages

Following are reasons to consider not using a single file pool server for BFS data:

- Administrators and system programmers who concern themselves with the operation of the entire system will have more production file pools to control and maintain. This can increase the time and effort associated with the maintenance and administration of the system.
- This setup could be confusing to users. The more file pool machines, the greater the risk of not finding your data.
- With the BFS, every user has access to a file pool, even if they are not enrolled.
- If you are using the OpenEdition Shell and Utilities Feature for VM/ESA to work in the BFS, you may need to install it again in each new file pool, or make an external link to a pool that has it installed. This should be *carefully documented*.

4.5 Shell and Utilities Feature Installation

The installation of the shell requires the successful completion of the following preparation steps:

- The BFSROOT EXEC has completed successfully.
- The DIRPOSIX utility has been run against the CP directory, or the POSIXINFO statements have been added as outlined in section 6.3.1, “First Time Handling of the Directory Source File” on page 141.
- The 2VMVMZ10 user ID with a GID and UID of 0 is defined.
- The C runtime library is available and at the latest service level.
- All file pool servers are started in multiple user mode and are available.
- A link to the MAINT 193 is established to access the LOADBFS utility.

From this point forward, the install consists of just a few VMSES procedures that are clearly outlined in the *Program Directory for OpenEdition Shell and Utilities Feature for VM/ESA* that was shipped with your product tape. After the install,

the product files can be inspected by accessing the disks using the VMFSETUP 2VMVMZ10 SHELL command from the 2VMVMZ10 user ID.

SHELL LOADBFS File

The installation uses a control file similar to the one used by BFSROOT. It is called the SHELL LOADBFS file and creates the following directories in the BFS:

```
/bin
/etc/samples
/usr/lib/nls/charmap
/usr/lib/nls/locale
/usr/lib/nls/locale/IBM
```

The objects loaded in `/bin` are the modules for the utilities included with the shell and the shell object. In the `/etc/samples` directory there is a `comics.lst` file that is entertaining. It can be also used to learn about the shell. The other directories will contain files required for national language support.

OPENVMDEFAULTS File

When the command OPENVM SHELL is run, it attempts to GLOBAL all the LOADLIBs that are needed to run the shell. This is, by default, the SCEERUN LOADLIB. It is recommended that this LOADLIB be stored on the MAINT 19F disk where it is accessible to all users.

The OPENVM SHELL command provides a way to customize the location and name of the libraries that are required for start up. This is done with a BFS control file called `/etc/openvmdefaults`. The following is a list of valid statements for the `openvmdefaults` control file:

CLIBNAMES *loadlib*
: You may have any number of additional or replacement LOADLIBs specified on the CLIBNAMES statement.

CLINKNAME *nickname*
: The value *nickname* is defined in a VMLINK NAMES file. This is the recommended way to manage nonstandard libraries.

CLINKNAME *userid vdev*
: The value *userid* identifies the user ID and *vdev* identifies the virtual device that OPENVM SHELL will link and access to obtain access to the C library

CLINKNAME .DIR *dirname*
: If the C library resides on the SFS, the directory specified by *dirname* will be accessed.

Below is an example of a VMLINK NAMES file that defines the MAINT 191 as the disk to link if the nickname CLIB is used. You may decide to use another disk better suited for public linking than MAINT 191.

```
00001 :nick.CLIB     :product.* *
00002                :title.C Library
00003                :node.TOTVM1
00004                :category.ALL
00005                :list.MNT191
00006
00007 :nick.MNT191   :product.MAINT 191
00008                :title.C library
00009                :node.TOTVM1
00010                :category.ALL
00011 * * * End of File * * *
```

Below is the contents of the /etc/openvmdefaults BFS file that uses the above nickname.

```
00001 CLIBNAMES SCEERUNL
00002 CLINKNAME CLIB
```

If the library specified is found on the currently accessed disks, the nickname or disk will *not* be accessed. When the disk is linked and accessed, it will be released and detached when the shell is exited from. Any libraries GLOBALed prior to when the OPENVM SHELL command is issued will remain part of the GLOBAL LOADLIB library list. Any additional libraries needed for the shell will be removed from the library list once the shell is exited from.

VMLINK has a large number of options. One use of them could be to invoke a preprocessing exit that further customizes your shell environment.

Hints and Tips

- **BFSROOT EXEC**

  Before you can install the OpenEdition Shell and Utilities, you must initialize the BFS by executing the BFSROOT EXEC (described in 4.4.6, “Create a Customized BFS” on page 88).

- **OpenEdition Shell and Utilities Modules**

  During the VMFINS BUILD step of installing the shell, many modules are generated. There are a few of them that match CMS commands or may match synonyms that you have already created. If you are working from the 2VMVMZ10 user ID, make sure not to issue any of the following commands until you are in the OpenEdition Shell environment.

  For example, entering ID will not result in a CMS IDENTIFY if the build disk (2VMVMZ10 49E) is accessed, because there is a MODULE called ID which will override it. This module is a shell command that will not work correctly in a CMS environment.

```
AWK      MODULE  SORT    MODULE  MKDIR   MODULE  ENV     MODULE
HEAD     MODULE  CKSUM   MODULE  TIME    MODULE  PASTE   MODULE
RM       MODULE  LOGGER  MODULE  DATE    MODULE  UNIQ    MODULE
BASENAME MODULE  STRIP   MODULE  MKFIFO  MODULE  EXPR    MODULE
ICONV    MODULE  CMP     MODULE  TOUCH   MODULE  PATHCHK MODULE
RMDIR    MODULE  LOGNAME MODULE  DD      MODULE  WC      MODULE
BC       MODULE  STTY    MODULE  MKNOD   MODULE  FALSE   MODULE
ID       MODULE  COMM    MODULE  TR      MODULE  PAX     MODULE
SED      MODULE  LP      MODULE  DIFF    MODULE  XARGS   MODULE
CAT      MODULE  SU      MODULE  MV      MODULE  FIND    MODULE
JOIN     MODULE  CP      MODULE  TRUE    MODULE  PR      MODULE
SH       MODULE  LS      MODULE  DIRNAME MODULE  YACC    MODULE
CHGRP    MODULE  TAIL    MODULE  NEWGRP  MODULE  FOLD    MODULE
LEX      MODULE  CPIO    MODULE  TSMAIL  MODULE  PRINTF  MODULE
SHBLTIN  MODULE  MAILX   MODULE  ECHO    MODULE  GETCONF MODULE
CHMOD    MODULE  TAR     MODULE  NOHUP   MODULE  PS      MODULE
LN       MODULE  CUT     MODULE  TTY     MODULE  GREP    MODULE
SLEEP    MODULE  MAKE    MODULE  ED      MODULE  PWD     MODULE
CHOWN    MODULE  TEE     MODULE  OD      MODULE
LOCALE   MODULE  C89     MODULE  UNAME   MODULE
```

- **Space Considerations in the BFS**

  After the execution of the BFSROOT EXEC, make sure that there is enough space available for creating the directories for the shell under ROOT. There should be at least 7500 free blocks available.

- **Service Level**

  To have the most trouble free environment, make sure your LE/370 libraries have the most recent service on them. The programs that comprise the shell are C applications.

4.6 Creating User File Pool Spaces

After you generate the file pool and the BFS server is running in multiple user mode, the file pool is ready for use. However, its organization forces BFS users to use file spaces created for users ROOT, TMP, and VAR.

You need to create file spaces for each of your BFS users who need to record data. It is also possible to create a file space for a department of users. Creating these additional file spaces lets you manage and confine the users′ data separately from system data.

Use the following command to enroll your users in the file pool and give them their own file spaces.

ENROLL USER *userid filepoolid* (BLOCKS *blks* STORGROUP *storgrp*
BFS USER *owning_userid* GROUP *owning_groupname*

Note: The above example is split into two lines due its length.

Hints and Tips

- When defining BFS file spaces for a user, if you omit the BFS parameter, then you will give the user a conventional CMS SFS space instead of the intended BFS file space.
- All the code needed by the administrator machine resides on the MAINT 193 minidisk, so it is required. If you are using an existing machine for the administration of the file pool, remember to access this disk.
- A server can contain a BFS file space named "X," or an SFS space named "X," but not both.

To use these additional file spaces, external links or additional mounts will be required. These topics are discussed in section 4.9.1, "External Links" on page 99 and section 4.10, "Mount Points" on page 100.

4.7 Adding Space to a File Pool Server

Once you have the server in operation and the users start to create data, you may need to increase the amount of space. Also, you may find that a file pool server will require additional space to hold the Shell and Utilities code. The disk may be increased by:

1. Define, attach, and CMS format (using 4K block size) the minidisk that you want to add on the server virtual machine.
2. Create a file to identify the new disk. This file will be similar to the POOLDEF file for the server, and it will be added to the POOLDEF during the process of adding the minidisk to the file pool. An example of this file is shown in Figure 55.

```
DDNAME=MDK00005       VDEV=308    GROUP=3
```

Figure 55. File Contents Used to Add Disk to Server

3. Once you have the definition file ready, use the command FILEPOOL MINIDISK to add this minidisk to the file pool. The format of the command is:

   FILEPOOL MINIDISK *serverid fname ftype fmode filepoolid* (NOFORMAT

   fname ftype fmode is the name of the file that contains the file pool minidisk definition. You also can use the FORMAT option in the FILEPOOL MINIDISK command, but if the file pool is very busy at the moment you are adding this minidisk, you could degrade the performance of the file pool.

Hints and Tips

Do not define a virtual disk in storage or a temporary disk as a file pool minidisk. You could lose data if you lose these defined minidisks.

4.8 Restoring the Environment after Installation Errors

If all the installation steps complete with no errors, there is no need to restore the environment. However, if there are any problems during the installation, this section gives you an idea of what happened during the specific installation steps, what EXECs and tables are involved, and how to restore the original environment.

The following information is based on having created a customized file pool server configuration exclusively for BFS data. If you chose the defaults for installation of the BFS, some of this section will not apply.

4.8.1 Errors during BFS Server Generation

If there are any problems during the generation of the BFS server, please ensure that your statements in the POOLDEF and DMSPARMS files are correct. Once the errors are corrected you can issue the FILESERV GENERATE command again. For a complete description of the available options during the generation of a BFS, see *VM/ESA CMS File Pool Planning, Administration, and Operation*, SC24-5751. To make sure that there is no data left in the BFS, we recommend you format all the minidisks (except the 191) from the BFS server before you regenerate the BFS.

Note: This step is not recommended for an already existing BFS, only one generated new for BFS data.

4.8.2 Restoring the BFS after the Execution of BFSROOT

Using the BFSROOT EXEC generates the initial set of objects and directories needed for the shell. The control file that contains all the definitions for the BFS is named BFS LOADBFS. This file normally resides on the MAINT 193 minidisk. If you customized it, it may have been copied to another disk.

To restore the environment, make sure that the right BFS, as defined by the MOUNT identifier in the BFS LOADBFS table, is mounted. After that, you can enter the corresponding command, as shown in Table 9, for the matching statements in the BFS LOADBFS file, starting from the bottom upwards. It is important to do this in the reverse order. Included on the diskette is the UNLOADBF EXEC, to perform this task, as discussed in section 4.8.4, “LOADBFS Tables.”

4.8.3 Restoring the BFS after Building the Shell File Trees

Many file trees are built during the installation of the shell. The files for the shell are copied into the BFS exactly the same as the base BFS objects. The configuration table that is used to generate the BFS is called SHELL LOADBFS and resides on the 2VMVMZ10 2B2 minidisk.

4.8.4 LOADBFS Tables

Table 9 on page 97 shows the different statements used in the BFS and SHELL LOADBFS tables and the corresponding command to restore the original environment.

Table 9. Equivalent Commands for the LOADBFS Tables	
Syntax in the LOADBFS table	**Necessary command to restore BFS**
DEFAULT_OWNER ...	no action required
DEFAULT_OWNER ...	no action required
DEFAULT_GROUP ...	no action required
DEFAULT_PERMISSION ...	no action required
MOUNT ...	no action required. This is the identifier for the file pool ID you have to mount in order to issue the OPENVM commands.
UNMOUNT ...	no action required
ENROLL *userid*... (*filepoolid*...	DELETE USER *userid filepoolid*
DIR *dirname*...	OPENVM ERASE *dirname*
EXTLINK *pathname*...	OPENVM ERASE *pathname*
SLINK *pathname*...	OPENVM ERASE *pathname*
MKNODE *pathname*...	OPENVM ERASE *pathname*
FILE *pathname*...	OPENVM ERASE *pathname*

There is no doubt that this is a lot of work. By using a unique BFS server for the shell, you can skip all these commands and regenerate the BFS server. However, in a production environment, you may be sharing the BFS with regular SFS users. In this case there is no option left but to execute these commands, or use the UNLOADBF EXEC located on the enclosed diskette. The code for this EXEC is shown in Figure 112 on page 196 in Appendix B, “Program Source Code” on page 183. Its function is to:

- Read the contents of the LOADBFS control file requested
- Select the records containing the keywords shown in Table 9
- Create the proper command strings
- Output these commands to an executable file

Figure 56 on page 98 shows an extract of the output file BFS UNLOADBF, which is created by invoking: *UNLOADBF BFS*.

```
00001 /*-------------------------------------------*/
00002 /* Output from execution of UNLOADBF BFS     */
00003 /* Change as required and run as an EXEC.    */
00004 /* If there is a 'MOUNT' required, check the  */
00005 /* bottom of this file and move or execute   */
00006 /* this mount as required.                   */
00007 /* Note: Must have proper permissions in     */
00008 /* order to run these commands.              */
00009 /*             Rundate: 07 Jul 1996 11:08    */
00010 /*--------------------------------- SG24-4747 */
00011 Address 'CMS'
00012 'OPENVM ERASE /usr/include/sys/time.h'
00013 'OPENVM ERASE /dev/null'
00014 'OPENVM ERASE /dev/tty'
00015 'OPENVM ERASE /usr/lib/nls/msg/C'
00016 'OPENVM ERASE /usr/lib/nls/msg/En_US'
00017 'OPENVM ERASE /usr/spool'
00018 'OPENVM ERASE /usr/mail'
00019 'OPENVM ERASE /u'
...
00040 'OPENVM MOUNT /../VMBFS:VMSYS:ROOT/ /'
00041 'DELETE USER VAR VMSYSU'
00042 'DELETE USER TMP VMSYSU'
00043 'DELETE USER ROOT VMSYS'
```

Figure 56. Extract from File BFS UNLOADBF Created by UNLOADBF EXEC

Using the UNLOADBF EXEC will enable you to use any LOADBFS control file that you may have modified to do a BFS backout, without having to manually enter the commands. The EXEC does not actually remove the objects, but creates an output file with the commands that do so. This makes it easy to review and see if what the EXEC thought it should do is actually what you want to do.

4.9 A BFS Supported by Multiple File Pool Servers

When the installation starts to grow, the complexity of the system grows too. That forces administration of the resources to become more complex.

There may come a time when you need to distribute your BFS file system over various file pool servers to avoid conflicts, to create file trees that connect merging departments' functions, and to give users excellent response time.

In the ideal setting, the VMSYS and VMSYSU file pools contain all of the production data. Users are enrolled in a different file pool server with their own BFS file space. This file space is mounted on /home directory either by an OPENVM MOUNT command or a mount external link in the file system. Static file trees go in VMSYS, and variable file trees go in VMSYSU.

Defining the server for user data is an identical process as described in section 4.4, "A New File Pool for BFS Data" on page 80. The process will lead you to the point where you will have more than one BFS server. We used the procedures to create a dedicated BFS server. The virtual machine is called VMSERBFS and the file pool name is VMBFS.

Figure 57 on page 99 shows the test file pool configuration.

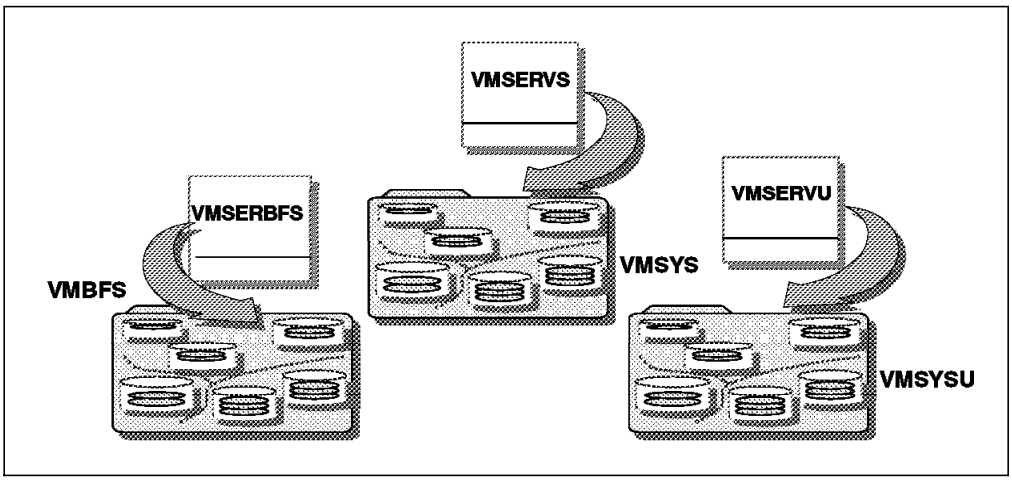


Figure 57. Dual BFS Server

4.9.1 External Links

If you use another BFS server and you have installed OpenEdition Shell and Utilities Feature for VM/ESA, you must decide how to access it. You can install the OpenEdition Shell and Utilities Feature for VM/ESA in the new file pool, or access it through an external link. The same applies to your profile and user files.

The external link has the following types associated with it:

CMSEXEC This link indicates that the file to which the external link is being created is an executable file on a minidisk or accessed SFS directory. The referenced executable file must have a file type of MODULE. DCE uses this function to access its code.

CMSDATA This link indicates the file to which the external link references are opened by C run-time library (C RTL) `fopen()` routine when the external link is opened. (in other words, this allows you to access CMS data residing in a minidisk.)

MOUNT This indicates that the external link is a mount external link (MEL). When a MEL is encountered during path name resolution, it is treated like a directory with a file system mounted on it; path resolution continues in the ″mounted″ BFS.

CODE This is a keyword that indicates the external link is in an application-defined format. This allows you to use the external link as part of your application.

The link is created with the OPENVM CREATE EXTLINK command. A complete set of options is listed in *OpenEdition for VM/ESA: Command Reference*, SC24-5728.

Creating an External Link to Shell and Utilities Code

In order to make the shell and utilities code available to a new BFS server, create an external link with this command:

```
OPENVM CREATE EXTLINK /../VMBFS:VMBFS:ROOT/bin MOUNT /../VMBFS:VMSYS:ROOT/bin
```

Hints and Tips

- A target mount external link (MEL) must be a directory.
- Note that the target object does not have to exist. That is, the object that contains the pointer to the code that is needed does not have to exist.
- You may create a external link that in turn references another external link. However, a maximum of eight levels of nesting is allowed.

The next step uses the already-defined profile and other tools residing in the VMSYS file pool, specifically in the /etc directory. Use the following command to create the link:

```
OPENVM CREATE EXTLINK /../VMBFS:VMBFS:ROOT/etc MOUNT /../VMBFS:VMSYS:ROOT/etc
```

Figure 58 shows the external links between the file pools.

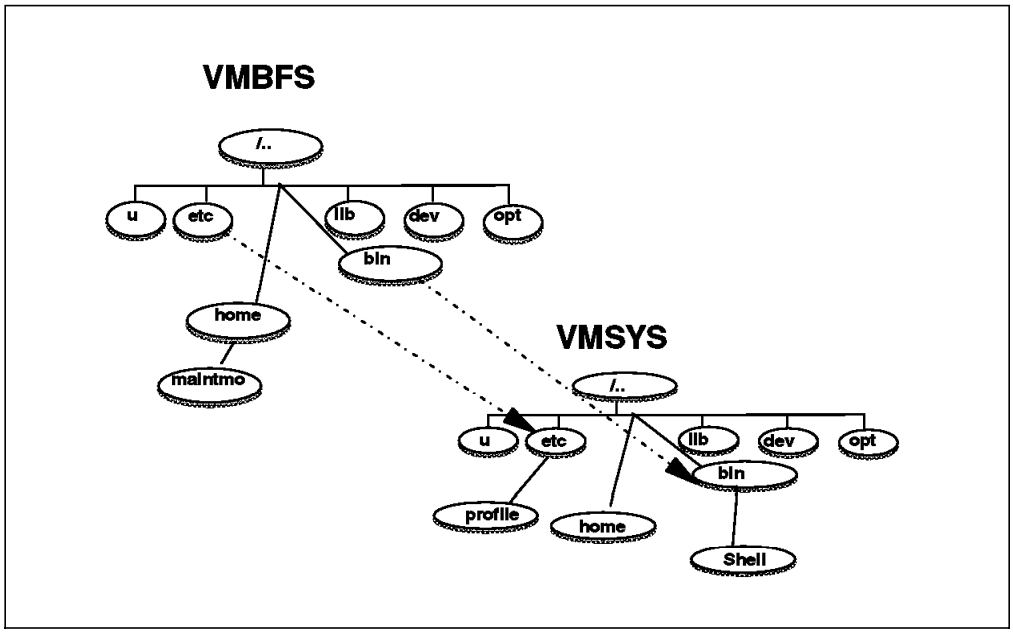


Figure 58. External Links Providing Shell Access to New File Pool

4.10 Mount Points

Since the BFS is a POSIX implementation, you have the structure of the UNIX file system. In it you have the concept of a mount file system. A mount file system is a file tree that has its own root point (not to be confused with the / directory). These files systems are associated with file spaces in the file pool server.

The file system is attached to the user's ID and it is accessible for the usual operations of read, write, and execute. The place at which the mounted file system's root directory is attached is known as the *mount point*. UNMOUNT the file system to detach this structure from the initial point. The OPENVM MOUNT

and UNMOUNT commands create and delete the addressability of each file space to a user.

It is through the combination of several mount commands that several file spaces can be combined in a way that extends the scope of the file system for the user, and allows the administrator better system controls.

Once you have defined a file space for a user or group of users, it can overlay a portion of any other file system. After the file system is mounted, its top-level directory takes the name of the directory on which it was mounted. In fact, the root directory of a mounted file system replaces the directory on which the file system is mounted.

As a collateral effect, any files that were originally in the mount directory disappear when the new file system is mounted, and thus they cannot be accessed. They will reappear when the file system is unmounted.

The next sections contain a few examples where parts of the BFS file tree are replaced with other trees using a second mount point. Keep in mind that the OPENVM MOUNT command explicitly mounts a file system. An external link can be used to implicitly mount file trees, and this is recommended.

4.10.1 User Home Directories Using External Links

If you have a production system and want to have user files and subtrees stored in file spaces apart from the production ones, you can give each user a BFS file space (as described in section 4.6, “Creating User File Pool Spaces” on page 94) and couple them to the main file tree with a MEL.

Figure 59 is an example of a simple mount point where the root file space is mounted as root. This is the most common form of mounted file system.

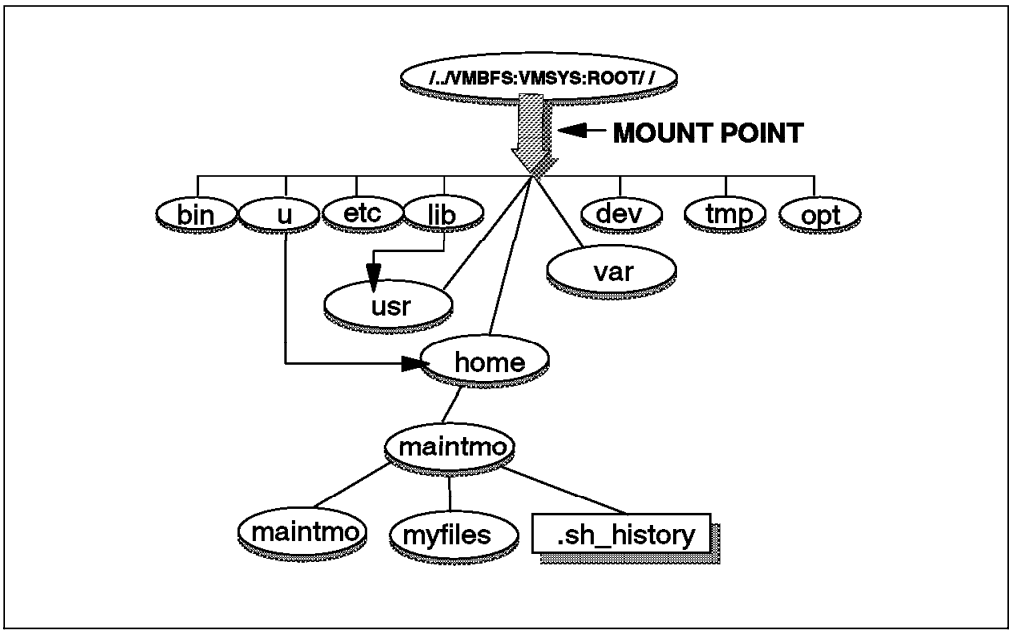


Figure 59. Standard Mount Point

You can determine your current mount point is with the command OPENVM QUERY MOUNT. Figure 60 on page 102 is an example of this command.

```
OPENVM QUERY MOUNT
Mount point = '/'
Type  Stat  Mounted
BFS   R/W  '/../VMBFS:VMSYS:ROOT/'
Ready; T=0.01/0.01 10:26:04
```

Figure 60. OPENVM MOUNT Command Example

Creating a Mount External Link

To attach a user BFS file space to a directory tree so that the user file space is implicitly mounted when the user enters their home directory, use the following command:

```
OPENVM CREATE EXTLINK /../VMBFS:VMSYS:ROOT/home/maintmo2 MOUNT /../VMBFS:VMBFS:MAINTMO/
```

When the user changes into the directory /home/maintmo2 the file space MAINTMO is mounted and all the directories and files within it will be available.

Figure 61 shows the user mount point.

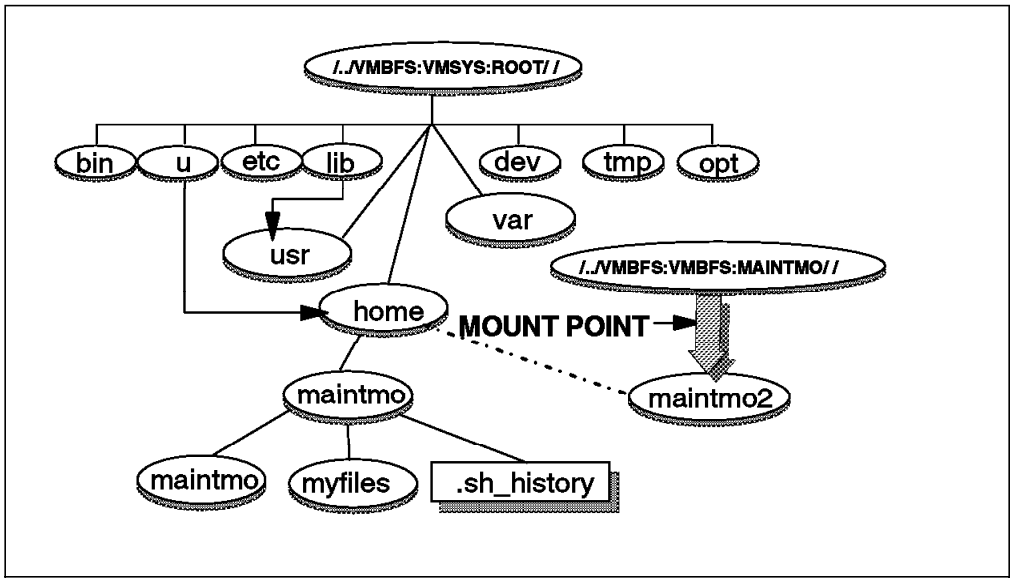


Figure 61. User Mount Point Example

Figure 62 shows the result of a `ls -l` shell command showing the mount point.

```
$
ls -l
total 0
drwxrwxrwx   1 maintmo  system     0 Apr 19 15:00 maintmo
Erwxrwxrwx   1 maintmo  system    24 Apr 19 11:05 maintmo2 -> /../VMBFS:VMBFS:MAINTMO
$
```

Figure 62. Shell ls Command Example Showing External Links

Figure 63 on page 103 lists the series of OPENVM commands needed to check what is mounted, mount the user mount point, and obtain information about the newly created mount point.

```
OPENVM QUERY MOUNT
Nothing is mounted
Ready; T=0.01/0.01 11:06:30

OPENVM MOUNT /../VMBFS:VMBFS:MAINTMO/ /
Ready; T=0.01/0.01 11:06:36

OPENVM QUERY MOUNT
Mount point = '/../ VMBFS:VMSYS:ROOT/home/maintmo2'
Type  Stat  Mounted
BFS   R/W  '/../ VMBFS:VMBFS:MAINTMO/'
```

Figure 63. OPENVM LIST Command Example

4.10.2 Temporary User Mount Points

As a permanent solution, you can create an external link by using the OPENVM CREATE EXTLINK command. This allows you to implicitly extend a file tree.

You can also get the same results by combining OPENVM MOUNT commands. Figure 64 shows the commands required to mount additional file spaces to a root file tree.

```
OPENVM MOUNT /../VMBFS:VMSYS:ROOT/ /
Ready; T=0.01/0.01 10:34:44                                    1

OPENVM Q MOUNT                                                 2
Mount point = '/'
Type  Stat  Mounted
BFS   R/W  '/../ VMBFS:VMSYS:ROOT/'
Ready; T=0.01/0.01 10:34:47

OPENVM LIST /                                                  3
Directory = '/'
Update-Dt  Update-Tm Type  Links          Bytes Path name component
04/19/1996 14:53:05   E3       -              - 'bin'
04/15/1996 10:20:42   D        -              - 'dev'
04/15/1996 10:20:39   D        -              - 'etc'
04/19/1996 11:17:36   D        -              - 'home'
04/15/1996 10:20:41   L        -              - 'lib'
04/15/1996 10:20:39   D        -              - 'opt'
04/15/1996 10:20:40   E3       -              - 'tmp'
04/15/1996 10:20:41   L        -              - 'u'
04/15/1996 10:20:40   D        -              - 'usr'
04/15/1996 10:20:40   E3       -              - 'var'
Ready; T=0.01/0.02 10:34:52

OPENVM MOUNT /../VMBFS:VMBFS:MAINTMO/ /home/maintmo/
Ready; T=0.01/0.01 10:35:15                                        4

OPENVM Q MOUNT                                                     5
Mount point = '/ home/maintmo'
Type  Stat  Mounted
BFS   R/W  '/../ VMBFS:VMBFS:MAINTMO/'
Mount point = '/'
Type  Stat  Mounted
BFS   R/W  '/../ VMBFS:VMSYS:ROOT/'
```

Figure 64. OPENVM MOUNT and Mount Point Flexibility

1 The current file pool is mounted. From here you can access all normal structures.

2 This is the root mount point.

3 This shows the contents of root.

4 Using the OPENVM MOUNT command, another file space is mounted to overlay an existing part of the root file tree (or extend it if no objects exist at the mount point).

5 Looking for what is mounted, you can see that the actual mount point is /home/maintmo and it is mounted in the root mount point of the /../VMBFS:VMSYS:ROOT/

```
OPENVM LIST /                                                          6
Directory = '/'
Update-Dt  Update-Tm Type  Links          Bytes Path name component
04/19/1996 14:53:05   E3       -              - 'bin'
04/15/1996 10:20:42   D        -              - 'dev'
04/15/1996 10:20:39   D        -              - 'etc'
04/19/1996 11:17:36   D        -              - 'home'
04/15/1996 10:20:41   L        -              - 'lib'
04/15/1996 10:20:39   D        -              - 'opt'
04/15/1996 10:20:40   E3       -              - 'tmp'
04/15/1996 10:20:41   L        -              - 'u'
04/15/1996 10:20:40   D        -              - 'usr'
04/15/1996 10:20:40   E3       -              - 'var'
Ready; T=0.01/0.02 10:35:18

OPENVM LIST /home/maintmo                                              7
Directory = '/home/maintmo'
Update-Dt  Update-Tm Type  Links          Bytes Path name component
04/30/1996 10:32:26   F        1            113 '.sh_history'
04/30/1996 10:10:22   F        1              0 'test.file
04/19/1996 15:37:09   E3       -              - 'bin'
04/19/1996 09:33:33   D        -              - 'home'
Ready; T=0.01/0.02 10:35:30

OPENVM Q MOUNT                                                         8
Mount point = '/home/maintmo'
Type  Stat  Mounted
BFS   R/W  '/../VMBFS:VMBFS:MAINTMO/'
Mount point = '/'
Type  Stat  Mounted
BFS   R/W  '/../VMBFS:VMSYS:ROOT/'
Ready; T=0.01/0.01 10:35:45

OPENVM UNMOUNT /home/maintmo                                           9
Ready; T=0.01/0.01 10:35:55

OPENVM Q MOUNT                                                        10
Mount point = '/'
Type  Stat  Mounted
BFS   R/W  '/../VMBFS:VMSYS:ROOT/'
Ready; T=0.01/0.01 10:36:04

OPENVM LIST /home/maintmo                                             11
Directory = '/home/maintmo'
Update-Dt  Update-Tm Type  Links          Bytes Path name component
04/30/1996 10:20:21   F        1           1197 '.sh_history'
04/16/1996 17:03:07   D        -              - 'maintmo'
04/16/1996 17:04:38   D        -              - 'myfiles'
Ready; T=0.01/0.02 10:36:10
```

Figure 65. OPENVM MOUNT and Mount Point Flexibility (Continued)

6 If you list your working root directory, you find that you still have the same *structure* as before the second mount, although there are some differences.

7 If you list the contents of your mount point, you will see the files that you already have in your mount point `/../VMBFS:VMBFS:MAINTMO/ /`. For testing the environment, we create a file call `test.file`. Notice that now you have a new `.sh_history` also.

8 Recheck the actual environment.

9 Unmount the second file space, using the same name that was used when it was mounted. This will disallow access to the files created when the second structure was mounted.

10 Check the actual mount point. You again have the top root file system mounted, `/../VMBFS:VMSYS:ROOT/`

11 The default set of files is shown after the replacement home file tree is unmounted.

As you see, this option allows you to have a separate file space for your own use, avoiding the need for an external link.

Chapter 5. System Administration

System administration is one of those jobs that if it is done well no one will thank you, but if it is done improperly every user on the system will complain. This section will help to keep the phone from ringing by giving you some examples on how to create a manageable POSIX environment.

5.1 Profiles for the Shell

Making the POSIX shell environment easier for users will make administration easier. Simple environment changes, such as displaying the current directory in the shell prompt, will make the environment easier to work in.

The POSIX shell profiles `profile` and `.profile` play a major role in the shell's life, just as their counterparts SYSPROF EXEC and PROFILE EXEC play a major role in the life of CMS itself.

5.1.1 System Profile

`profile` resides in the `/etc` directory and sets the environment for all users who share a file system. Typically this file is used to set system-wide variables, such as command aliases, search paths for command resolution, and shell prompts.

The OpenEdition Shell environment includes a sample profile located in `/etc/profile.sample`. Use the POSIX `cp` command to copy this file to a working profile:

```
cp /etc/profile.sample /etc/profile
```

Figure 66 on page 108 shows an example of a customized shell system profile.

```
# +-------------------------------------------------------------+
# | /etc/profile                                      (SG24-4747) |
# +-------------------------------------------------------------+
#
# set and export time zone (TZ) across shells
export TZ=EST5EDT,m4.1.0,m10.5.0
#
# ask CMS to clear the screen
cms 'VMFCLEAR'
#
# Issue date command with specific formatting
date '+%A %B %Od %r %Z %Y'
#
# %A  = full weekday name
# %B  = full month name
# %Od = day of the month numeric
# %r  = time in am - pm notation
# %Z  = time zone name
# %Y  = the year
#
# set and export the search path for command resolution
export PATH=/bin:/usr/bin:/etc:$HOME:$HOME/bin
#
# remember the old command prompt, to use with the new prompt
export OLDP=$PS1
#
# set the main prompt symbol (PS1) to the highlighted login name ($LOGNAME)
# and include the current directory ($PWD)
#    +---------+-----< This character has the value X'1D'
#    |         |       (Display shell user ID highlighted)
#    v         v
PS1='Y$LOGNAME-$PWD/ $OLDP '
alias dir="ls -al"              # Set dir  (similar to OS2)
alias list="ls -al"             # Set list to resemble CMS LISTfile
alias type="cat"                # Set type to resemble CMS TYPE
alias help="cms help oshell"    # Set help to go to CMS
alias q="cms q"                 # Set q    to go to CMS (QUERY)
#
# +-------------------------------------------------------------+
# | this portion taken as is from supplied /etc/profile.sample    |
# +-------------------------------------------------------------+
#
# The MAILMSG will be printed by the shell every MAILCHECK seconds
# (default 600) if there is mail in the MAIL system mailbox.
export MAIL=/usr/mail/$LOGNAME
export MAILMSG="You have new mail."
if [ -s "$MAIL" ]           # This is at Shell startup.  In normal
then echo "$MAILMSG"        # operation, the Shell checks
fi                          # periodically.
cms 'SET OUTPUT AD ['
cms 'SET OUTPUT BD ]'
cms 'SET INPUT [ AD'
cms 'SET INPUT ] BD'
```

Figure 66. Example of /etc/profile

The aliases are defined to allow a user to enter several CMS commands, which in turn execute the appropriate UNIX commands. The SET INPUT/OUTPUT commands translate special characters correctly.

5.1.2 User Profile

The .profile BFS file resides in the user's home directory and is used to further tailor the user environment. This is where changes can be made to the default system setup. Typical additions are:

- Additional command aliases
- Shell tmout value
- Override any setting in /etc/profile

VM/ESA CMS users may consider adding command aliases that allow them to enter additional CMS command names that actually perform the shell commands. This will help with the transition from CMS user to POSIX shell user. Of course, it may confuse the user if there are too many aliases.

Figure 67 is an example of a user .profile:

```
# +-----------------------------------------------------------+
# | My user profile                                           |
# +-----------------------------------------------------------+
#
alias fl="ls -al"              # Allow CMS filelist command to
                               # issue shell LS command
alias re=mv                    # Allow CMS rename command to
                               # issue shell move command
alias er="rm -i"               # Allow CMS erase command to
                               # issue shell remove command with
                               # additional safety prompt
TMOUT="120"                    # Set shell timeout to close shell
                               # after 2 minutes of no activity
                               # TMOUT must be in upper case
function xedit {               # Define a function to call XEDIT
  cms "XEDIT '$1' (NAMETYPE BFS"
  return                       # This could also be made into a
  }                            # separate xedit shell script
```

Figure 67. Example of User .profile

5.1.3 Profile Variables

Profile variables are used to customize the POSIX shell environment. They allow the system administrator to set a consistent look and feel between the VM/ESA CMS and POSIX shell environments. This is achieved with the use of profile variables such as command aliases.

Command Alias

Command aliases in the POSIX shell environment can be particularly helpful to the CMS user by allowing familiar commands to work in the shell environment. For example, a CMS user would use the ERASE command to delete a file while a UNIX user would use rm. By setting up an alias for rm, the new shell user will be able to use the more familiar ERASE command as well as rm. Figure 67 shows some aliases for the CMS user working in the POSIX shell.

Time Zones

As with all computer systems there is a choice of what definition is used for system time. An offset from coordinated universal time (UTC) may be used to set the VM/ESA system clock.

If the system clock is set on local time then there is no need for the `timezone` shell environment variable to be used; however, this does not happen very often.

If, however, the VM/ESA system clock is set to UTC, then the timezone (TZ) shell environment variable must be set with the appropriate offsets to reflect local time in the shell environment. The TZ variable can be found in `/etc/profile`.

The TZ variable format is:

[**std**][**offset**][**dst**][**offset**]**,rule**

These variables are defined as follows:

std Is the name of the time zone.

offset Is the offset.

dst Is the name of the time zone for summertime.

offset Is the offset for daylight savings time.

rule Indicates what day and time to change times.

TZ variables are discussed in more detail in the following sections:

std and **dst**

Only std is required. If dst is missing, daylight savings time does not apply. Uppercase and lowercase letters are allowed.

Any character can appear in the std or dst field except for the ones listed below (because the meanings of these letters and characters are unspecified): a leading colon (:), digits, comma (,), minus (-), plus (+), or the null character.

offset

Indicates the value one must add to the local time to arrive at Coordinated Universal Time (UTC). The offset has the form:

[**hh**[**mm ss**]]

The minutes (mm) and seconds (ss) are optional. The hour (hh) is required and may be a single digit. The hour must be between 0 and 24; minutes and seconds, if present, between 0 and 59.

An offset preceded by a minus (-) indicates a time zone east of the Prime Meridian. A plus (+) preceding offset is optional and indicates the time zone west of the Prime Meridian. The offset following std is required. If no offset follows **dst**, summer time is assumed to be 1 hour ahead of standard time. The difference between standard time offset and summer time offset must be greater than or equal to 0.

rule

Indicates when to change to and back from summer time. The rule format is:

[**date/time, date/time**]

where the first date describes when the change from standard to summer time occurs and the second date describes when the change back happens. Each time field describes when, in current local time, the change to the other time is made.

The format of date must be one of the following:

Jn The Julian day n (1-n-365). Leap days are not counted. That is, in all years, including leap years, February 28 is day 59 and March 1 is day 60. It is impossible to explicitly refer to the occasional February 29.

n The zero-based Julian day (0-n-365). Leap days are counted, and it is possible to refer to February 29.

Mm.n.d The day d (0-d-6) of week n (1-n-5) of month m (1-m-12) of the year. Day zero is Sunday. Week 1 is the first week in which the d′th day occurs. Week 5 is the week which the last d′th day in month m occurs; this may be either the fourth or fifth week, but it is always denoted as week 5.

The time has the same format as **offset** except that no leading sign, minus (-) or plus (+), is allowed. The default, if time is not given, is 02:00:00.

If **dst** is specified and the rule is not specified by TZ then the default for the summer time start date is M4.1.0 and for the summer time end date is M10.5.0.

If the TZ variable is not set, time conversions behave as if TZ were set to TZ=UTC.

Examples of How to Set the TZ Environment Variable: The value of TZ for the east coast of the United States is EST5EDT. EST5 is Eastern Standard Time, five hours west of UTC0. EDT is Eastern Daylight Time, which is understood to be four (5 minus 1).

The value of TZ for the east coast of Australia is EST-10EDT. EST-10 is Eastern Standard Time, ten hours east of UTC0. EDT is Eastern Daylight Time which is understood to be eleven (10 plus 1).

Shell History File

The POSIX shell automatically saves a list of commands entered. This list is stored in a file in the user′s home directory. The name of this file is set by the shell variable HISTFILE and is by default .sh_history. The default is to keep the last 128 commands. In most cases, the default should be adequate for most users. The default can be altered with the shell variable HISTSIZE. Maintain a size to suit your users, if you have too large a value then the response to shell commands will degrade.

5.1.4 Shell Prompts

The default shell prompt is a $ for a standard user or a # for a superuser. The prompt can be changed easily to something significant for users. This change is made in the /etc/profile.

The following section of code comes from the sample in Figure 66 on page 108.

```
# remember the old command prompt, to use with the new prompt
export OLDP=$PS1
#
# set the main prompt symbol (PS1) to the highlighted login name ($LOGNAME)
# and include the current directory ($PWD)
#    +---------+-----< This character has the value X'1D'
#    |         |       (Display shell user ID highlighted)
#    v         v
PS1='Y$LOGNAME-$PWD/ $OLDP '
```

Figure 68. Example of /etc/profile

The code shown in Figure 68 produces the following prompt for USER111:

user111 /home/user111/ $

This prompt, from left to right, contains the following information:

- The user name, which is highlighted
- The current directory the user is in
- The standard $ prompt, which is $OLDP

To highlight the user name, the string $LOGNAME is preceded by X'1DE8' and followed by X'1D60'. There are no spaces between $LOGNAME and the hex characters, and each two-byte highlight will take only one position on the display. This highlighting technique will not work in every environment, for example, it does not work in full-screen CMS.

5.1.5 Online Help Alias

New users to the OpenEdition Shell environment may find themselves referring to the online help often. Access to CMS HELP is directly available from the POSIX shell by the alias help="cms help oshell". CMS and CP commands can be passed from the POSIX shell directly to CMS by prefixing the commands with cms.

5.2 File System Space Monitoring

Consumption of storage in BFS storage spaces is a task that must be performed regularly. There are several considerations regarding the BFS and storage space:

- The threshold for a BFS file space is set to 100%. There is no warning that a storage space is becoming full.
- Users share file spaces (ROOT, TMP, VAR) for mail and temporary files. When these file spaces become full, unpredictable results occur. It is extremely difficult to detect that the problem is a file space problem not a C runtime problem.

- A BFS file tree can be split over many servers and file spaces creating a difficult environment to monitor.
- There may be several BFS file systems to monitor.
- BFS file spaces do not permit temporary storage consumption in excess of the file space limits. Therefore, BFS storage consumption is always limited to the current maximum number of blocks set for the target BFS file space. Because BFS file spaces cannot directly contribute to the temporary storage excesses, they do not directly participate in the storage use exits designed to handle the problem.

The BFSPACE EXEC, located on the enclosed diskette, will assist you in monitoring the file space consumption that is part of a file tree. This EXEC is a skeleton that can tailored to your specific system's requirements. The output of this EXEC is shown in Figure 69.

```
bfspace
Looking for EXTERNAL links, symbolic links not reported

Server: VMSYS      Directory: /
Userid    Storage Group  4K Block Limit  4K Blocks Committed  Threshold
ROOT              2               29000          8089-27%      100%

Server: VMBFS      Directory: /home/vetter/
Userid    Storage Group  4K Block Limit  4K Blocks Committed  Threshold
MAINTSV           2                  20             18-90%      100%

Server: VMSYSU     Directory: /tmp/
Userid    Storage Group  4K Block Limit  4K Blocks Committed  Threshold
TMP               2               11000             22-00%      100%

Server: VMSYSU     Directory: /var/
Userid    Storage Group  4K Block Limit  4K Blocks Committed  Threshold
VAR               2               31000         11262-36%      100%
Ready; T=1.15/1.41 15:15:21
```

Figure 69. BFSPACE Utility to Monitor BFS File Spaces

From the output of the BFSPACE EXEC, you can see that the MAINTSV file space is becoming full. The file space should be given more blocks, or unused data should be cleared from it.

5.3 File System Backup and Restores

A system administrator must arrange for the backup of all mission-critical system data on a regular basis. The OpenEdition Byte File System (BFS) is no exception. The BFS is controlled by the Shared File System (SFS) which has built-in backup and restore functions. The system administrator can use these integrated SFS functions to back up and restore the BFS.

As with normal SFS operation, there are two types of data in a file pool: user data and control data. The backup and recovery of control data is covered in the *VM/ESA CMS File Pool Planning, Administration and Operation*, SC24-5751. There are no special considerations for the backup and recovery of control data for the BFS.

5.3.1 File System Backup Considerations

VM/ESA Version 2 Release 1.0 introduced two new commands to assist in the backup and migration of BFS data. They are FILEPOOL UNLOAD and FILEPOOL RELOAD. Though the old SFS backup commands still exist and work with BFS data, take a look at the new commands. They perform a function similar to FILESERV MOVEUSER without the need to stop the server, and without the restriction that the file space be moved within the same file pool. They also perform the same operation as FILEPOOL RESTORE, except there is no longer the restriction that the storage group minidisk configuration remain the same between backup and restore.

Note: The following sections are not the only backup and restore topics in this publication. Refer to 2.12, “File Transfer and Archive” on page 53 for additional backup ideas using file transfer programs.

File System Backup Methods

There are two standard file pool commands that can be used for backing up user data. They are FILEPOOL BACKUP and FILEPOOL UNLOAD. Table 10 contains a comparison of the two backup methods.

Table 10. File Server Backup and Restore Methods

Backup Method	Restore Individual Files	Restore BFS Files	List Method of Backup	Backup Storage Group	Backup File Space
FILEPOOL BACKUP/RESTORE	Using FILEPOOL FILELOAD	By inode	FILEPOOL LIST BACKUP	√	
FILEPOOL UNLOAD/RELOAD	Using FILEPOOL RELOAD FILES	By inode and by fully-qualified path name	FILEPOOL LIST BACKUP	√	√

The FILEPOOL LIST BACKUP command will list the fully qualified path name by default. It will list both the fully qualified path name and the CMS short name by using the all-object option ALLOBJ. Figure 75 on page 120 is an example of FILEPOOL LIST BACKUP and its output.

For ease of use when restoring individual user files, it is recommended to use the FILEPOOL UNLOAD/RELOAD backup strategy, with fully qualified path names.

In order for you to associate a file referenced by its CMS short file name with its pathname, you need to record which BFS files have what CMS short file name (inode).

File System Locks

During file system backup the backup machine will attempt to obtain a *share disable lock* for the storage group or file space being backed up. A share disable lock allows users to read from the storage group or file space, but does not allow them to make updates. This has the potential to cause problems with applications that do event logging. Some applications may need to be shut down before BFS backups are initiated.

To help resolve file lock issues during BFS backup, the SET FILEWAIT ON command should be used. The SET FILEWAIT ON command is entered before the FILEPOOL BACKUP or FILEPOOL UNLOAD commands. If FILEWAIT ON is not used and a user other than the user ID running the backup is reading from or writing to the file space or storage group, the FILEPOOL BACKUP and FILEPOOL UNLOAD commands will terminate. When FILEWAIT is ON, the file pool server does not terminate the backup if it cannot immediately obtain the lock. Instead, the file pool server does not let any new activity start and waits for all activity to end before obtaining the locks for the backup process.

Applications competing for the filepool should also SET FILEWAIT ON to prevent backups and other locks from disrupting their process.

At the completion of the FILEPOOL BACKUP or FILEPOOL UNLOAD, the file pool server automatically releases the file pool lock.

There are several strategies that can be used to help identify the user that may be causing a locking problem.

- The QUERY FILEPOOL CONFLICT command can be used to display information about lock conflicts in a specified file pool. This information helps to diagnose why the file pool is locked.

  The QUERY FILEPOOL CONFLICT must be issued for the user that is trying to obtain the lock. Figure 70 shows user MAINT is trying to obtain a share disable lock for a file pool backup but user RODNASH is already holding a lock on the file pool. Here the backup MAINT has initiated will not start until the RODNASH releases the lock.

```
query filepool conflict maint vmsys:
Requestor  Holder    Wait            Lock           Lock Type  Identifier
MAINT      RODNASH   I/O             File_Space     IExcl      000200000000000
Ready; T=0.01/0.01 11:53:52

query filepool conflict rodnash vmsys:
Requestor  Holder    Wait            Lock           Lock Type  Identifier
RODNASH              None
Ready; T=0.01/0.01 11:54:41
```

Figure 70. Example of QUERY FILEPOOL CONFLICT Command

- There is a little-known document on the MAINT 193 minidisk called SFSDOT LIST3720. The commands described in the document are intended to be used under the direction of IBM Service. SFSDOT contains a series of commands to help an SFS administrator resolve problems.

  The .SHOW LOCK USER and the .SHOW LOCK WANTLOCK commands can be used to identify which user is holding a lock on the file pool and what type of lock it is.

  These commands must be issued from the file pool server virtual machine, not the user ID initiating the backup.

  Figure 71 on page 116 is the same scenario as in Figure 70 but this time the .SHOW LOCK commands are used to display the lock contention. The first two commands show that RODNASH still has a lock on a resource. The last command shows that MAINT (running the backup) has requested a lock.

```
.show lock user .bfslock
         LOCK                                    NUMB
USERID   TYPE  SIX IS  IX  S   X   CS  CU  CX  WAIT  LOCK-ID
RODNASH  GROUP 0   0   1   0   0   0   0   0   0    0002             CHKB
RODNASH  UOID  0   0   1   0   0   0   0   0   1    0002000000000001 CHKB
DMS5BC3065I Operator command processing complete

.show lock user rodnash
         LOCK                                    NUMB
USERID   TYPE  SIX IS  IX  S   X   CS  CU  CX  WAIT  LOCK-ID
RODNASH  GROUP 0   0   1   0   0   0   0   0   0    0002             CHKB
RODNASH  UOID  0   0   1   0   0   0   0   0   1    0002000000000001 CHKB
DMS5BC3065I Operator command processing complete

.show lock wantlock *
          LOCK                      REQUEST  REQUEST
USERID    TYPE   LOCK-ID            STATE    MODE     DURATION
MAINT     UOID   0002000000000001   G WAIT   CS       CHKOUT
DMS5BC3065I Operator command processing complete
```

Figure 71. Example of .SHOW LOCK Commands

5.3.2 Concurrent File System Backup

To minimize the impact of BFS backups, backups of different storage groups or file pools can be run concurrently from more than one administration machine. Although the file pool server still has the same amount of work to perform as if the backup were done sequentially from a single administration machine, the overall amount of time for the backup could be reduced. Overlapping I/O across multiple storage groups or file pool minidisks could also significantly reduce the amount of time required for the backup.

5.3.3 File System Backup Examples

In the following two examples, both FILEPOOL UNLOAD and FILEPOOL BACKUP will be used. As FILEPOOL BACKUP only works at the storage group level and FILEPOOL UNLOAD works at both the storage group and file space level, FILEPOOL UNLOAD will be used at the file space level and FILEPOOL BACKUP at the storage group level.

For systems with only a small number of file spaces, backing up by file space offers greater flexibility than a total file pool backup. For a larger system with many file spaces, backing up by storage group is far more manageable. It is far easier to restore one or two storage groups than to restore several hundred file spaces.

The default BFS consists of three file spaces, namely ROOT, TMP, and VAR. This lends itself to a file space backup. However, adding additional file spaces for users is recommended. To simplify server backups, a carefully managed file system could segregate BFS file spaces from SFS file spaces by storage group or by server.

The FILEPOOL administrator tools reside on the MAINT 193 minidisk. This disk must be accessed when issuing FILEPOOL UNLOAD/RELOAD or FILEPOOL BACKUP/RESTORE commands.

File System Backup Using FILEPOOL UNLOAD

To back up a user storage group or file space using the FILEPOOL UNLOAD command, follow these steps:

1. Log on to a user ID with administration authority for the server containing BFS data. This data may be spread out over many servers.

   Note: Supplied with this publication is a tool that helps to determine all of the file spaces that hold the data for a complete BFS directory tree.

2. LINK and ACCESS the MAINT 193 minidisk to access the FILEPOOL commands.
3. Enter a FILEDEF command to identify the backup file, which can be on either tape or disk.
4. Enter SET FILEWAIT ON
5. Enter a FILEPOOL UNLOAD command, such as:

   FILEPOOL UNLOAD FILESPACE ROOT

6. At the completion of the backup, enter SET FILEWAIT OFF

These steps can be automated in a small EXEC.

Figure 72 is an example of a file pool backup using FILEPOOL UNLOAD.

```
acc 193 t
Ready; T=0.01/0.01 18:02:06

set filewait on
Ready; T=0.01/0.01 18:02:15

filedef unload disk root unload w
Ready; T=0.01/0.01 18:02:48

filepool unload filespace root
DMSWFP3485I FILEPOOL processing begun at 18:03:53 on 9 Apr 1996.
DMS5PX3438I FILEPOOL UNLOAD successful
DMSWFP3486I FILEPOOL processing ended at 18:04:29 on 9 Apr 1996.
Ready; T=0.96/2.44 18:04:29

set filewait off
Ready; T=0.01/0.01 18:05:15
```

Figure 72. File Pool Backup Using FILEPOOL UNLOAD

File System Backup Using FILEPOOL BACKUP

To back up a user storage group using the FILEPOOL BACKUP command, follow these steps:

1. Log on to a user ID with administration authority for the server containing BFS data. A BFS file tree may be spread over several storage groups and file pool server machines.
2. LINK and ACCESS the MAINT 193 minidisk to access the FILEPOOL backup tools.
3. Have the read passwords ready for the storage group minidisks, because FILEPOOL BACKUP will link to each minidisk in the storage group. An alternative is to have LNKNOPAS as a directory option for this user ID. If

you use the link password approach, take time to review the CP directory and change the link passwords from the installation default values (if required).

4. Enter a FILEDEF command to identify the backup file, which can be on either tape or disk.
5. Enter SET FILEWAIT ON
6. Enter a FILEPOOL BACKUP command, such as:

   FILEPOOL BACKUP 2

7. At the completion of the backup, enter SET FILEWAIT OFF

Figure 73 is an example of a file pool backup using FILEPOOL BACKUP.

```
acc 193 t
Ready; T=0.01/0.01 17:01:49

filedef backup disk stg2 backup w
Ready; T=0.01/0.01 17:00:00

set filewait on
Ready; T=0.01/0.01 17:00:16

filepool backup 2
DMSWFP3485I FILEPOOL processing begun at 17:01:54 on 9 Apr 1996.
DMSCRL1136E Unable to gain access to library FSMINT
DMS5PR3593I BACKUP file creation begun for storage group 2
DMS5PR3593I in file pool VMSYS at 17:01:54 on 04/09/96.
DMS5PR3533I Linking to minidisk MDK00002 at 0305 as 0FFF.
DASD 0FFF LINKED R/O; R/W BY VMSERVS
DMS5PR3533I Linking to minidisk MDK00003 at 0306 as 0FFE.
DASD 0FFE LINKED R/O; R/W BY VMSERVS
DMS5PR3533I Linking to minidisk MDK00004 at 0307 as 0FFD.
DASD 0FFD LINKED R/O; R/W BY VMSERVS
DMS5PR3502I Backing up the catalog data. Time = 17:01:54.
DMS5PR3502I Backing up minidisk MDK00002. 27147 total data blocks remain
DMS5PR3502I to be backed up. Time = 17:01:57.
DMS5PR3502I Backing up minidisk MDK00003. 20322 total data blocks remain
DMS5PR3502I to be backed up. Time = 17:02:17.
DMS5PR3502I Backing up minidisk MDK00004. 12469 total data blocks remain
DMS5PR3502I to be backed up. Time = 17:02:40.
DASD 0FFF DETACHED
DASD 0FFE DETACHED
DASD 0FFD DETACHED
DMS5PR3635W Backup file not registered with DFSMS/VM
DMS5PR3500I Backup of storage group 2 in file pool VMSYS
DMS5PR3500I successfully completed at 17:03:18 on 04/09/96.
DMSWFP3486I FILEPOOL processing ended at 17:03:19 on 9 Apr 1996.
Ready(00004); T=1.29/4.38 17:03:19

set filewait off
Ready; T=0.01/0.01 17:04:16
```

Figure 73. File Pool Backup Using FILEPOOL BACKUP

Note: Message DMSCRL1136E and the return code of 4 in Figure 73 are issued because the DFSMS/VM code was not accessed at the time. DFSMS/VM is not necessary for the successful execution of the FILEPOOL BACKUP/RESTORE commands.

5.3.4 Restoring Individual Files

The following sections contain experiences regarding file restoring. These experiences gained while working with the BFS.

Locating File Names to Restore

To identify which file a user wants restored, it may be necessary to produce a list of the files that were backed up. A listing of these files can be produced using the FILEPOOL LIST BACKUP command.

As with all FILEPOOL commands, the FILEPOOL LIST BACKUP command resides on the MAINT 193 minidisk. This minidisk must be accessed before entering the command.

The FILEPOOL LIST BACKUP command requires two FILEDEF commands to be entered. One FILEDEF command is used as input and points to the backup file on either tape or disk. The other FILEDEF command is used for output and defines where the output is written. Figure 74 is an example of FILEPOOL LIST BACKUP command.

```
acc 193 t
Ready; T=0.01/0.01 18:17:23

filedef backup disk root unload w
Ready; T=0.01/0.01 18:18:12

filedef listbkup disk rootunld report w
Ready; T=0.01/0.01 18:18:55

filepool list backup filespace root (allobj
DMSWFP3485I FILEPOOL processing begun at 18:19:26 on 9 Apr 1996.
DMS5PT3438I FILEPOOL LIST BACKUP successful
DMSWFP3486I FILEPOOL processing ended at 18:19:27 on 9 Apr 1996.
Ready; T=0.20/0.21 18:19:27

listfile rootunld report w
FILENAME FILETYPE FM FORMAT LRECL       RECS     BLOCKS   DATE     TIME
ROOTUNLD REPORT   W1 V        185       1500         27  4/09/96 18:19:27
Ready; T=0.01/0.01 18:19:47
```

Figure 74. Example of FILEPOOL LIST BACKUP Command

Figure 75 on page 120 is an example of the output created by the FILEPOOL LIST BACKUP command.

```
Time: 18:19:25                    Filepool List Backup - VMSYS
Date: 04/09/96                    Listing Selection    - FILESPACE

Backup File Created:
    04/09/96 AT 22:03:53



Type CMS short name Path name
---- -------------- ---------
D                   /../VMBFS:VMSYS:ROOT/
F    1BE      0     /../VMBFS:VMSYS:ROOT/.sh_history/
D    1C       0     /../VMBFS:VMSYS:ROOT/bin/
D    2        0     /../VMBFS:VMSYS:ROOT/dev/
D    3        0     /../VMBFS:VMSYS:ROOT/etc/
D    4        0     /../VMBFS:VMSYS:ROOT/home/
S    161      0     /../VMBFS:VMSYS:ROOT/krb5/
S    13       0     /../VMBFS:VMSYS:ROOT/lib/
F    4BF      0     /../VMBFS:VMSYS:ROOT/lst/
D    5        0     /../VMBFS:VMSYS:ROOT/opt/
E    11       0     /../VMBFS:VMSYS:ROOT/tmp/
```

Figure 75. Example of Output Created by FILEPOOL LIST BACKUP Command

It is worth noting that the CMS short name or inode number of a file, when displayed with the FILEPOOL LIST BACKUP command or the OPENVM LIST command, is always displayed in hexadecimal. When inode numbers are displayed from within the POSIX shell environment using `ls -i`, they are displayed in decimal.

The backup file created by FILEPOOL BACKUP and FILEPOOL UNLOAD has a creation date and time that is expressed in UTC. This can be seen in Figure 75.

File Restore Using FILEPOOL RELOAD FILES

To restore a user file from a storage group or file space using the FILEPOOL RELOAD FILES command (backed up using UNLOAD), follow these steps:

1. Log on to a user ID having administration authority for the servers containing the BFS data you wish to restore.
2. LINK and ACCESS the MAINT 193 minidisk to access the FILEPOOL utilities.
3. Create a CMS file named CONTROL RELOAD, which contains a list of files to be restored.

   The files can be specified in CMS short name or fully qualified path names. File specification cannot be mixed; use only one type in the control file.

   The CMS short name would look like this:

   ```
   5C9      0        VMSYS:root
   ```

   The fully qualified path name would look like this:

   ```
   /../VMBFS:VMSYS:ROOT/home/user111
   ```

4. Enter a FILEDEF command to identify the backup file.
5. Enter the restore command:

   ```
   FILEPOOL RELOAD FILES VMSYSU
   ```

Figure 76 on page 121 is an example of restoring a file using the FILEPOOL RELOAD FILES command.

```
acc 193 t
Ready; T=0.01/0.01 14:31:25

xedit control reload
Ready; T=0.01/0.01 14:31:39.

type control reload
/../VMBFS:VMSYS:ROOT/home/user111/test4
Ready; T=0.01/0.01 14:31:45

filedef reload disk stg2 unload w
Ready; T=0.01/0.01 14:32:36

filepool reload files vmsys
DMSWFP3485I FILEPOOL processing begun at 14:32:47 on 10 Apr 1996.
DMS5PY3594R Files from file space ROOT in file pool VMSYS
DMS5PY3594R will replace files in file pool VMSYS from reload file
DMS5PY3594R created on 04/10/96 at 18:29:33
DMS5PY3594R Enter '1' to continue or '0' to cancel.

1
DMS5PY3617I File successfully restored: /../VMBFS:VMSYS:ROOT/home/rodnash/test4
DMS5PY3620I 1 files restored
DMS5PY3438I FILEPOOL RELOAD successful
DMSWFP3486I FILEPOOL processing ended at 14:32:59 on 10 Apr 1996.
Ready; T=0.41/0.74 14:32:59
```

Figure 76. Example of FILEPOOL RELOAD FILES Command

File Restore Using FILEPOOL FILELOAD

To restore a user file from a storage group using the FILEPOOL FILELOAD command, follow these steps:

1. Log on to a user ID with administration authority for the file pool server containing the BFS file to be restored.
2. LINK and ACCESS the MAINT 193 minidisk to access the FILEPOOL commands.
3. Create a CMS file named CONTROL FILELOAD, which contains a list of files to be restored.

   The files must be specified in CMS short names.

   The CMS short name would look like:

   ```
   5C9      0        VMSYS:root
   ```

4. Enter a FILEDEF command to identify the backup file.
5. Enter the restore command:

   ```
   FILEPOOL FILELOAD 2 VMSYSU (ASIS
   ```

Figure 77 on page 122 is an example of restoring a file using the FILEPOOL FILELOAD command:

```
acc 193 t
Ready; T=0.01/0.01 14:14:15

xedit control fileload
Ready; T=0.01/0.01 14:14:22

type control fileload
5C9 O VMSYS:root
Ready; T=0.01/0.01 14:14:50

filedef restore disk stg2 backup w
Ready; T=0.01/0.01 14:15:17

filepool fileload 2 vmsys (asis
DMSWFP3485I FILEPOOL processing begun at 14:15:52 on 10 Apr 1996.
DMS5PS3617I File 5C9 O VMSYS:ROOT. successfully restored
DMS5PS3620I 1 files restored
DMSWFP3486I FILEPOOL processing ended at 14:16:54 on 10 Apr 1996.
Ready; T=0.33/0.69 14:16:54
```

Figure 77. Example of FILEPOOL FILELOAD Command

5.4 DFSMS and the Byte File System

In VM/ESA, storage management is simplified, efficient, and centrally controlled. This advancement in storage management is because of the Shared File System and Data Facility Storage Management Subsystem for VM (DFSMS/VM). The SFS eliminates wasted space and DASD fragmentation (a small number of cylinders left between minidisks). It allows an installation to assign a specified amount of storage space to a user. Unlike the minidisk concept, this space is not allocated to a user until the space is actually required for storing data. DFSMS/VM helps minimize the amount of DASD containing inactive SFS files by keeping only active SFS files on the primary DASD, migrating inactive files to secondary storage and erasing files that have met expiration criteria.

To DFSMS/VM, the BFS looks like any other SFS file data and can be managed in the same way. The big advantage that the VM/ESA implementation of POSIX has over several UNIX environments, is that it is able to migrate and compress inactive files to other DASD levels or to tape. You could, for example, regularly move all shell history files to a BFS directory that is managed by DFSMS/VM and that will be purged it is mode than 7 days old.

The process DFSMS/VM uses to manage a BFS environment is similar to how DFSMS/VM manages an SFS environment. The difference is in what DFSMS/VM sees as the enrolled user in a file pool.

In a standard BFS installation, only three users are enrolled in the file pool. They are ROOT, TMP and VAR. All directory structures typically begin with ROOT, but may extend to mount several other file spaces.

If you follow the recommendation in this book to define each user as their own mount point, then BFS administration is even easier and follows standard SFS administration. You add ROOT, TMP, and VAR to the list of file spaces dedicated for BFS user data.

An important difference in the way DFSMS/VM works with a BFS is that the Interactive Storage Management Facility (ISMF) cannot be used for some functions. Some ISMF functions require that the user directory being listed is accessed by the virtual machine ISMF is running on. A BFS user directory cannot be accessed in the traditional sense in which you can access an SFS user directory.

5.4.1 Inodes for CMS Files

DFSMS/VM does not see a file in a BFS by its fully qualified name, for example:

```
/../VMBFS:VMSYS:ROOT/home/user111
```

but by its CMS short name:

```
5C9      0        VMSYS:root
```

In any DFSMS/VM command where a file name and file type specification is used, the CMS short name or inode must be used. The OPENVM LIST command can be used to display the CMS short name of a file in the BFS.

5.4.2 DFSMS Installation Considerations

DFSMS/VM Function Level 221 was used to explore the migration capabilities of DFSMS/VM with the BFS. The maintenance level was 9501. No changes were necessary for the DFSMS/VM to support BFS data, further supporting the integration of VM/ESA POSIX with the standard VM/ESA environment.

When using DFSMS 221 with DIRMAINT Version 1 Release 5 make sure that you contact IBM level 2 support (for either DFSMS or DIRMAINT) and request APAR VM60499. This APAR allows DFSMS to use the new DIRMAINT 1.5 command structure.

5.4.3 Managing a BFS with DFSMS/VM

The following are the steps involved in managing a BFS with DFSMS/VM:

1. Authorize the file pool servers containing BFS data to DFSMS/VM.

   Use the normal procedure that would be used to authorize an SFS server. For example, update the DGTVAUTH DATA control file.

2. Change the file pool server to be DFSMS-managed.

   Use the normal procedure to allow an SFS file pool server to be DFSMS/VM managed. For example, update the VMSERVS DMSPARMS control file.

3. Define a management class or classes for the BFS.

   The management classes currently in use for an SFS could be used for a BFS. It is likely that you will want to implement the same policies in both environments. There is no reason to treat BFS data any differently from SFS data.

   Decide the management class or classes to use; either define new management classes or existing management classes.

4. Use the DFSMS ALTER command to set a management class for each mount point (file space).

   ```
   DFSMS ALTER * * VMBFS:ROOT MAN BFS ( NOWAIT
   ```

   The format of the DFSMS ALTER is:

   - File name and file type

- File mode or directory ID
- Management class
- Management class name

5. Use the DFSMS MANAGE command to initiate file migration:

```
DFSMS MANAGE STO 2 VMBFS: ( NOTHRES NOWAIT
```

The format of the DFSMS MANAGE command is:

- Storgroup
- Group number
- File pool ID

Figure 78 on page 125 is an example of the migration process using the DFSMS ALTER and MANAGE commands.

```
dfsms alter * * vmbfs:root. man bfsnow (nowait
DGTUDR2026I DFSMS request 480 accepted for processing
Ready; T=0.01/0.02 14:32:16
RDR FILE 0133 SENT FROM SMSMASTR PUN WAS 0094 RECS 0147 CPY  001 A NOHOLD NOKEEP

receive 133 alter listing a File ALTER LISTING A1 created from DFSMS001 T0000480 B1
Ready; T=0.01/0.03 14:32:43

type alter listing a
DFSMS Function Level 221                          04/19/96   14:32:16
DFSMS ALTER with NOWAIT, request identifier 480

DFSMS ALTER processing started for * * VMBFS:ROOT.

The following files were assigned the new management class BFSNOW:

FILENAME  FILETYPE  OLD MANAGEMENT CLASS
--------  --------  --------------------
A0        0               BFS
C7        0               BFS
A1        0               BFS
A6        0               BFS
A7        0               BFS
1AB       0               BFS
1AD       0               BFS
...
1CA       0               BFS

     302 files matched the fileid specified
     302 files were eligible for processing
     302 files assigned the new management class

DFSMS ALTER with NOWAIT completed with no errors
Ready; T=0.03/0.14 14:34:19

dfsms manage sto 2  vmbfs: (nothres nowait
DGTUDR2026I DFSMS request 483 accepted for processing
Ready; T=0.01/0.02 14:35:41
RDR FILE 0134 SENT FROM SMSMASTR PUN WAS 0095 RECS 0016 CPY  001 A NOHOLD NOKEEP

receive 134 manage listing a File MANAGE LISTING A1 created from DFSMS001 T0000483 B1
Ready; T=0.01/0.02 14:37:22

t manage listing a
DFSMS Function Level 221                          04/19/96   14:35:41
DFSMS MANAGE with NOWAIT, request identifier 483

DFSMS MANAGE processing started for storage group 2 in file pool VMBFS

DFSMS MANAGE STORGRP command summary:

     Storage group percent utilization before MANAGE: 12
     Storage group percent utilization after MANAGE: 1
     10975 blocks available in primary storage group
     15648 blocks available in ML1
     0 files and directories converted
     0 files and directories could not be converted
     0 files occupying 0 4K blocks erased
     300 files eligible to be migrated
     300 files occupying 1353 4K blocks have been migrated from primary storage to ML1
     0 files occupying 0 4K blocks have been migrated from primary storage to ML2
     0 files occupying 0 4K blocks have been moved from ML1 to ML2

DFSMS MANAGE with NOWAIT completed with no errors
```

Figure 78. Example of BFS Migration

5.4.4 DFSMS/VM Reporting

After files from the BFS have been migrated to another storage level, DFSMS/VM can report on how many files have been migrated and what the space savings are. For reporting, use the DFSMS REPORT command:

```
DFSMS REPORT STO 2 VMBFS:
```

or

```
DFSMS REPORT SPA FILESP VMBFS:ROOT.
```

Figure 79 shows the output a DFSMS REPORT command.

```
dfsms report spa sto 2 vmbfs: DGTUDR2026I DFSMS request 486 accepted for processing
Ready; T=0.01/0.02 14:39:01

RDR FILE 0135 SENT FROM SMSMASTR PUN WAS 0096 RECS 0062 CPY  001 A NOHOLD NOKEEP
receive 135 report listing a

File REPORT LISTING A1 created from DFSMS001 T0000486 B1 received from SMSMASTR at TOTVM1
Ready; T=0.01/0.02 14:40:27
type report listing a
DFSMS Function Level 221                          04/19/96   14:39:01
DFSMS REPORT SPACEMANAGEMENT STORGROUP, request identifier 486

DFSMS REPORT SPACEMANAGEMENT STORGROUP processing started for file pool VMBFS and storage group 2

DFSMS REPORT SPACEMANAGEMENT FILESPACE processing started for file pool VMBFS and userid ROOT

Files migrated for this file space:                                  285
   Migration Level 1:                                 285
   Migration Level 2:                                   0
Logical 4K blocks currently in use by this file space:              1280
Physical 4K blocks in use by this file space:                        794
Physical 4K blocks in use by primary storage for this file space: -         42
Physical 4K blocks in use by secondary storage for this file space:        836
   Migration Level 1:                                 836
   Migration Level 2:                                   0
Physical 4K blocks saved for this file space due to compaction:      486

DFSMS REPORT SPACEMANAGEMENT FILESPACE processing started for file pool VMBFS and userid TMP

Files migrated for this file space:                                    3
   Migration Level 1:                                   3
   Migration Level 2:                                   0
Logical 4K blocks currently in use by this file space:                 7
Physical 4K blocks in use by this file space:                          5
Physical 4K blocks in use by primary storage for this file space: -          2
Physical 4K blocks in use by secondary storage for this file space:          7
   Migration Level 1:                                   7
   Migration Level 2:                                   0
Physical 4K blocks saved for this file space due to compaction:        2

DFSMS REPORT SPACEMANAGEMENT FILESPACE processing started for file pool VMBFS and userid VAR

Files migrated for this file space:                                   12
   Migration Level 1:                                  12
   Migration Level 2:                                   0
Logical 4K blocks currently in use by this file space:                18
Physical 4K blocks in use by this file space:                         16
Physical 4K blocks in use by primary storage for this file space: -          4
Physical 4K blocks in use by secondary storage for this file space:         20
   Migration Level 1:                                  20
   Migration Level 2:                                   0
Physical 4K blocks saved for this file space due to compaction:        2
```

Figure 79. Example of DFSMS REPORT

5.4.5 DFSMS/VM Recalling Files

Any migrated files that are subsequently referenced will be automatically decompressed and recalled to the original file pool server. Files can also be recalled by using the DFSMS RECALL command:

```
DFSMS RECALL 2C8 O VMBFS:ROOT. ( NOWAIT
```

Figure 80 shows the output the DFSMS RECALL command:

```
openvm list /etc (name header                    1
Directory = '/etc'
File name File type Byte file system  Type  Path name component
5B0       0         VMSYS:ROOT.       F    'bfs'
5E1       0         VMSYS:ROOT.       F    'count'
5D3       0         VMSYS:ROOT.       F    'du'
71        0         VMSYS:ROOT.       F    'mailx.rc'
22C       0         VMSYS:ROOT.       F    'profile'
73        0         VMSYS:ROOT.       F    'profile.sample'
1D        0         VMSYS:ROOT.       D    'samples'
6F        0         VMSYS:ROOT.       F    'startup.mk'
313       0         VMSYS:ROOT.       F    'x'
70        0         VMSYS:ROOT.       F    'yylex.c'
72        0         VMSYS:ROOT.       F    'yyparse.c'
5CC       0         VMSYS:ROOT.       F    'BFS'
Ready; T=0.02/0.02 15:29:21
cd /etc
 rodnash /etc/ $

ls -ils                                          2
total 152
 1484   8 -rwxrwxrwx   1 dceadmin system       15 Apr 10 16:07 BFS
 1456   8 -rwxrwxrwx   1 dceadmin system       15 Apr  2 10:33 bfs
 1505   8 -rwxrwxrwx   1 dceadmin system      250 Apr 11 16:58 count
 1491   8 -rwxrwxrwx   1 dceadmin system       63 Apr 11 17:02 du
  113   8 -rw-r--r--   1 bin      bin         1199 Sep 14  1995 mailx.rc
  556   8 -rw-rw-rw-   1 dceadmin system     1945 Apr 10 16:10 profile
  115   8 -rw-r--r--   1 bin      bin         654 Sep 14  1995 profile.s
   29   0 drwxr-xr-x   1 dceadmin system        0 Sep 14  1995 samples
  111  16 -rw-r--r--   1 bin      bin        4353 Sep 14  1995 startup.m
  787   8 -rwxrwxrwx   1 dceadmin system       29 Apr  2 10:30 x
  112  24 -rw-r--r--   1 bin      bin       11682 Sep 14  1995 yylex.c
  114  48 -rw-r--r--   1 bin      bin       20943 Sep 14  1995 yyparse.c
 rodnash /etc/ $
exit
Ready; T=0.01/0.02 15:32:01

dfsms recall 73 0 vmbfs:root. ( nowait DGTUDR2026I DFSMS request 495 accepted for processing
Ready; T=0.01/0.02 15:35:01
RDR FILE 0138 SENT FROM SMSMASTR PUN WAS 0096 RECS 0062 CPY  001 A NOHOLD NOKEEP

receive 138 recall listing a File REPORT LISTING A1 created from DFSMS001 T0000486 B1 received
Ready; T=0.01/0.02 15:35:29
type recall listing a
DFSMS Function Level 221                          04/19/96   15:35:01
DFSMS RECALL with NOWAIT, request identifier 495

DFSMS RECALL processing started for 73 0 VMBFS:ROOT.
     1 files matched the fileid specified
     1 files were eligible for processing
     1 files recalled from migration level 1
     0 files recalled from migration level 2
DFSMS RECALL with NOWAIT completed with no errors
```

Figure 80. Example of DFSMS RECALL

To identify the CMS short name or inode of a file for recall, use the OPENVM LIST command (1) or the POSIX `ls -ils` (2) command. The POSIX `ls -ils` command will display the inode number in decimal and the OPEMVM command

will display the inode in hexadecimal. The hexadecimal inode number must be used with the DFSMS RECALL command.

5.5 Scanning the Byte File System

The administrator of the POSIX environment should periodically review the BFS to look for specific objects, such as dumps. In the VM/ESA POSIX environment, dumps seem to be scattered about everywhere by the applications. Dumps from the POSIX shell environment are generally directed to /tmp, but dumps from some applications like DCE′s dced may be placed in their own directory structure.

Dumps are of concern to system administrators not only because they indicate abnormal system problems but also because they take up valuable space in the file system. If the system administrator does not search for dumps, they will stay in the file system forever and eventually fill the file spaces.

The administrator can use the shell′s find command in a simple shell script to find the dumps that are scattered throughout the file system.

There are several points to consider before using the find command:

- The -follow option should be used to make sure that all external mounts and symbolic links are followed during the search.

  This option is very useful since there are likely to be several external links in the file system configuration. For example, /tmp and /var in the default configuration are external links.
- This command should be run from a superuser to ensure that it can get to every directory in the file system.
- As the system file grows in size and complexity, this command could have a negative impact on system performance. Consider running the command during a quiet system time.
- These commands are helpful in finding any object, not just dumps.

The following command locates all dumps over 10 days old in the file system and places a listing of them in file called /tmp/results:

```
find / -name *dump*.* -o -name *DUMP*.* -mtime +10 -follow -type f -print >
/tmp/results
```

The following command performs the same function as the one above, but moves all the files into directory /tmp/dumps:

```
find / -name *dump*.* -o -name *DUMP*.* -mtime +10 -follow -type f -print
-exec mv {} /tmp/dumps \;
```

The following command performs the same function as the one above, but deletes the file after confirming the delete request:

```
find / -name *dump*.* -o -name *DUMP*.* -mtime +10 -follow -type f -print -ok
rm {} \;
```

Note: In the last two examples -exec and -ok are used to perform actions on the files that are found. find replaces the {} expression with the current path name

and then executes the resultant string as a shell command. In *OpenEdition for VM/ESA Command Reference*, SC24-5728, the `find` command, as discussed in the `-exec` and `-ok` sections, specifies that the semicolon (;) must be delimited by white space. In the examples, a backslash (\) is used as white space.

5.6 Running POSIX Applications in a Saved Segment

A saved segment is a range of pages of virtual storage you can define to hold data or re-entrant code. Defining frequently used data and code as saved segments provides several advantages, including:

- Several users can access the same common storage, so real storage usage is minimized.
- Using saved segments decreases the system I/O rate and DASD paging space, thereby improving system performance.

Saved segments allow code or data in an area of virtual storage to be saved and assigned a name. A saved segment can then be dynamically attached to, or detached from, a virtual machine by referencing this name.

Programs residing within the page ranges of a saved segment that are re-enterable can be shared by multiple virtual machines. This allows you to place code that is frequently referenced in a saved segment and load it into a virtual machine when needed.

In the VM/ESA POSIX environment, it is possible to save an application into a segment and then have a virtual machine load the segment for execution.

To explore this possibility, program object for the POSIX shell was used. It is up to the system administrator to determine which applications are best suited for storage in a saved segment.

5.6.1 POSIX Shell in a Saved Segment

The OpenEdition Shell does not make full use of the advantages of using saved segments, however the basic steps presented here can be reused to save other POSIX applications in a segment.

The steps required to save a POSIX application in a saved segment are as follows:

1. Log on to a user ID with CP privilege class E, and access to the BFS containing the shell code.
2. Copy the shell code from the BFS to a minidisk:

   ```
   OPENVM GETBFS /bin/sh SHELL MODULE A ( MODULE
   ```

3. Create a segment skeleton.

   ```
   DEFSEG OESHELL 2100-21FF EW
   ```

   Make sure that the segment being defined does not overlap any other segment that will be used by the virtual machine.
4. Create a physical segment file with entry for the logical segment.
5. Create a logical segment file with the shell module name.
6. Copy the SYSTEM SEGID file from the CMS system disk to the A-disk.

7. Generate the segment:

   SEGGEN OESHELL

8. Copy the updated SYSTEM SEGID file back to the CMS system disk.

   Figure 81 is an example of how to place the POSIX shell in a saved segment.

```
openvm getbfs /bin/sh shell module a (module
Ready; T=0.03/0.05 14:37:36

defseg oeshell 2100-21ff ew
HCPNSD440I Saved segment OESHELL was successfully defined in fileid 0081.
Ready; T=0.01/0.01 14:37:55

xedit oeshell pseg a
type oeshell pseg a
LSEG SHELL
Ready; T=0.01/0.01 14:38:15

xedit shell lseg a
type shell lseg a
MODULE SHELL ( SYSTEM SERVICE
Ready; T=0.01/0.01 14:38:46

acc 190 t
DMSACP725I 190 also = S disk
Ready; T=0.01/0.01 14:38:52

copyfile system segid s = = a (olddate type replace
DMSCPY721I Copy SYSTEM SEGID S2 to SYSTEM SEGID A2 (old file)
Ready; T=0.01/0.01 14:39:21

seggen oeshell
HCPNSS440I Saved segment OESHELL was successfully saved in fileid 0081.
Ready; T=0.01/0.05 14:39:36

q nss 81 map
FILE FILENAME FILETYPE MINSIZE  BEGPAG ENDPAG TYPE CL #USERS PARMREGS VMGROUP
0081 OESHELL  DCSS       N/A    02100  021FF   EW  A  00000   N/A       N/A
Ready; T=0.01/0.01 14:39:57

copyfile system segid a = = t (oldd ty rep
DMSCPY721I Copy SYSTEM SEGID A2 to SYSTEM SEGID T2 (old file)
Ready; T=0.01/0.01 14:40:25

acc 193 t
DMSACC724I 193 replaces T (190)
Ready; T=0.01/0.01 14:40:47

sampnss cms
HCPNSD440I The Named Saved System (NSS) CMS was defined in fileid 0082.
Ready; T=0.01/0.01 14:40:54

ipl 190 clear parm savesys cms
HCPNSS440I Named Saved System (NSS) CMS was successfully saved in fileid 0082.
VM/ESA V2.1.0   10/23/95 13:39

DMSACP723I M (300) R/O
Ready; T=0.07/0.09 14:41:15
```

Figure 81. Saved Segment Example, Setting Up the Segment

9. Load the segment into your virtual machine:

```
SEGMENT LOAD SHELL
```

10. Run the code from the segment

```
SHELL POSIX(ON) / -L
```

The OPENVM SHELL command cannot be used to invoke the code in the saved segment. The command OPENVM SHELL issues an explicit OPENVM RUN /bin/sh and would thus avoid the module saved in the segment.

The command SHELL POSIX(ON) / -L must be used when the code is in the saved segment. The POSIX(ON) option tells the C runtime library that this is a POSIX application. The -L option is used to start the shell as a login shell.

Figure 82 shows the final steps involved in running a POSIX application from a saved segment:

```
segment load shell
Ready; T=0.01/0.01 14:41:30

q segment
Space     Name      Location  Length    Loaded   Attribute
OESHELL   SHELL     02100000  00040858  YES      USER
CMSVMLIB  DMSRTSEG  0178F480  0003FBE0  YES      SYSTEM
CMSVMLIB  VMLIB     01700000  0008F440  YES      SYSTEM
CMSPIPES  PIPES     01900000  00079830  YES      SYSTEM
HELPINST  CMSINST   00700000  00056CD8  YES      SYSTEM
Ready; T=0.01/0.01 14:41:34

global loadlib sceerun
Ready; T=0.01/0.01 14:41:54

q loadlib
LOADLIB  = SCEERUN
Ready; T=0.01/0.01 14:42:07

shell posix(on) / -L
IBM
Licensed Material - Property of IBM
5654-030 (C) Copyright IBM Corp. 1995
(C) Copyright Mortice Kern Systems, Inc., 1985, 1993.
(C) Copyright Software Development Group, University of Waterloo, 1989.

All Rights Reserved.

U.S. Government users - RESTRICTED RIGHTS - Use, Duplication, or
Disclosure restricted by GSA-ADP schedule contract with IBM Corp.

IBM is a registered trademark of the IBM Corp.


Thursday April 11 02:41:57 PM EDT 1996
  /home/maint/ $

exit
```

Figure 82. Saved Segment Example: Running the Shell

In practice, putting the POSIX shell into a saved segment does not significantly improve storage usage or I/O reduction. The benefit lies in the fact that it

provides an example that it is possible to execute a complete POSIX application from a saved segment.

5.7 Data Sharing in a VM Collection

For VM systems that commonly share file resources through system interconnections, the following sections discuss these distributed environments regarding the BFS.

5.7.1 File Access in a Distributed Environment

If the BFS server and the user who wishes to access BFS files reside on different VM images, any of the three VM communications servers can provide the connectivity between the systems. However, the POSIX security data provided by CP and sent to the server may be different, depending on the communications server used.

If the systems are in a Communication Services (CS) or TSAF collection (that is, the systems are connected by and running the Inter-System Facility for Communication (ISFC) or the Transparent Services Access Facility (TSAF)), the POSIX security values provided are those of the POSIX process that initiated the connection. If the connection is with AVS (APPC/VM VTAM Support), the values provided are those of the security user at the target system.

5.7.2 File Processing in a Distributed Environment

If the BFS server and the user wishing to execute a set-ID file resides on two different VM images, those images must be in the same CS collection (that is, the images must be running and connected by the Inter-System Facility for Communications (ISFC)), and the system administrator must ensure that:

- There is a single name space for the UIDs and GIDs across the collection. That is, every UID value in the collection must represent a distinct user or set of users, as must every GID. A single CP user ID name space is also required.
- The server where the Byte File System resides is a global resource.
- All systems in the path between the two systems, and the server and requestor systems, must be running a version of CP that supports POSIX.

For more information, consult *VM/ESA: Planning and Administration*, SC24-5750.

5.8 API Overview

The introduction of POSIX support in OpenEdition for VM/ESA provides an internationally supported application program interface (API) through which POSIX-compliant applications are easily ported.

VM has a long history of guest support and software emulation. The UNIX function calls added in the POSIX support are the next extension to the operating system known for guest support (MVS and VSE) and OS simulation under CMS. The areas of LE/370 environment (C Libraries), OpenEdition callable services library routines, and the OpenEdition Byte File System are the major changes to VM/ESA required to support the POSIX implementation. Additional changes were made to CP.

5.8.1 LE/370 Environment

The POSIX programming interfaces 1003.1, 1003.1a, and 1003.1c are provided as C library routines in SCEERUN LOADLIB and SCEELKED TXTLIB, as part of the IBM Language Environment for MVS and VM C-Runtime that is shipped with VM/ESA Version 2.

The POSIX functions are available to any C/VM program compiled with the POSIX(ON) pragma, or compiler option. In addition, a language-neutral interface to a subset of the POSIX functions is provided to any CMS program through the nucleus-resident Callable Services Library (CSL), VMMTLIB. The CSL contains only a subset of the calls because some of the calls are implemented entirely in the C RTL without assistance from VMMTLIB.

At this time, the fork() set of calls is not provided, however a similar function spawn() is provided and is a proposed POSIX standard.

For application development, and to use the c89 shell utility, you must install the IBM C for VM/ESA compiler.

The C for VM/ESA code should be loaded to MAINT's 19E disk. The SCEERUN LOADLIB and the SCEELKED TXTLIB should be loaded there as well, so they are available to applications. However, if you tailor your system to put the C/VM runtime library files somewhere else, or split up the C-specific files from the Language Environment Common Execution Library files, you can also tailor the process by which the applications gain access to the runtime library. This is explained in section "OPENVMDEFAULTS File" on page 92.

As a performance improvement, you can load certain LE/370 routines into saved segments. Placing routines into saved segments reduces overall system storage requirements by making the routines shared, and execution times are reduced for each application, since load time decreases. All LE/370 routines that can be installed into saved segments are included in the CEEBLSPA and CEEBLSPB build list. These build lists contain a list of parts that implement the POSIX environment for LE/370. They will be used to construct the SCEE segment (to be loaded below the 16M line), and the SCEEX segment (to be loaded above the 16M line).

The LE/370 code that is shipped with VM/ESA Version 2 Release 1.0 is only the C code and the Common Execution Library (CEL) portions. The PL/1 and COBOL portions of the full LE/370 are *not* included. This code is necessary to run the OpenEdition Shell and Utilities Feature for VM/ESA, and to run any other compiled C application that uses LE/370 facilities. For more information about the Language Environment for VM/ESA, refer to *IBM Language Environment for MVS & VM, Concepts Guide*, GC26-4786.

5.8.2 OpenEdition Callable Services Library Routines

As an interface between the VM/ESA operating system and the functions specified in the POSIX standards, OpenEdition for VM/ESA provides access to a set of callable services known as the OpenEdition for VM/ESA Callable Services.

The OpenEdition CSL routines are provided to enable language run-time environments to implement POSIX functions, and to provide system programmers a language-neutral subset of POSIX.

OpenEdition for VM/ESA comes with three CSLs:

- **VMLIB** contains routines that:
  - Call SFS and minidisk functions.
  - Access REXX variables and issue CMS commands.
  - Manipulate the CMS program stack.
  - Call CPI-communications.

  VMLIB contains more than these examples, and this includes common routines that can be used across various IBM systems and VM extension routines.
- **VMMTLIB** contains routines for:
  - Multitasking
  - Tracing
  - Accounting
  - ABEND processing
  - OpenEdition VM/ESA services
- **POSXSOCK** contains routines for developing application programs that are portable across DCE platforms using OpenEdition for VM/ESA sockets. They are described in *IBM OpenEdition for VM/ESA: Sockets Reference*, SC24-5741.

5.9 Servicing OpenEdition for VM/ESA

Once the product is in production, updates to the source code may be made from the product laboratories. OpenEdition is updated using VMSES/E. If the product or component is in VMSES/E format you can apply service in one or more of the following forms.

- Program Temporary Fix (PTF)

  A temporary modification to a source code made by the product development group, which is the owner of the component.
- Corrective Service (COR)

  A single or collection of PTFs that will fix a specific problem. It can be delivered on a wide variety of media types.
- Product Service Update (PSU)

  The PSU is a procedure that refreshes the current service database and leaves customized files untouched. It does not back-level any reach-ahead service. This procedure uses the VM/ESA Recommended Service Upgrade (RSU), which has all recommended service pre-installed. This process saves time since each RSU is pre-applied and objects are already build.
- Local service

  Any local modification you wish to make must be applied using VMSES/E.

VM/ESA service is packaged on its delivery media at the component level. Each component has a separate source product parameter file, $PPF, that identifies the service for the component on the delivery media.

Table 11 on page 135 contains the names of the principal VM/ESA components and their $PPF identifications. In this list are included all the basic VM/ESA components as well as DCE, LE/370, and the Shell and Utilities feature.

Table 11. VMSES/E Component Description		
Component Name	**Component ID**	**Base $PPF File Name**
CMS	568411201	2VMVMA10
CP	568411202	2VMVMB10
AVS	568411204	2VMVMD10
REXX	568411205	2VMVMF10
TSAF	568411206	2VMVMH10
DV	568411208	2VMVMI10
VMSES	568411209	2VMVMK10
GCS	568411210	2VMVML10
SHELL	568411214	2VMVMZ10
DCEBASE	568411215	2VMVMG10
DCEPRIV	568411216	2VMVMS10
GUI	568411217	2VMVMY10
LE370	568819805	5688198E
ICKDSF	565899201	5684042H

For a complete description about the service process, refer to *VM/ESA Service Guide*, SC24-5749.

5.9.1 LE/370 and VM/ESA Version 2 Release 1.0

Included with every VM/ESA Version 2 system is a subset of the IBM Language Environment for MVS & VM Version 1.5.0, 5688-198. The SCEELKED, SCEERUN, CEE*, and EDC* files (and others) are part of this, along with the SCEE and SCEEX physical shared segments. Regarding these runtime and common libraries:

- C is the only supported language. These libraries do not include COBOL or PL/I support.
- Contact your IBM support center to order the most recent maintenance for this component. The Shell and Utilities feature, the DCE feature, and the POSIX support require current service levels. Failure to maintain the current service levels could result in CMS ABENDs.
- The C libraries are automatically loaded as needed by the OPENVM commands. You may customize this installation in the `openvmdefaults` file.
- In an environment where multiple users run POSIX applications, using the SCEE and SCEEX segments will have a positive impact on system performance.
- For POSIX application development, the IBM C for VM/ESA compiler must be purchased. It is the only compatible compiler for the `c89` shell utility.
- At the time of this publication, LFS is not able to run using LE/370 libraries. See the VM/ESA Version 2 Release 1.0 announcement letter for additional information.
- Your current software profile will determine which LE/370 service you receive (containing PL/I and COBOL C updates, or not), when you order service for 5688198E.

5.9.2 Maintenance User IDs Associated with OpenEdition

There are several new user IDs associated with OpenEdition for VM/ESA. The names of the user IDs are similar to the component IDs used for service. Table 12 contains a list of the user IDs used to perform service on the OpenEdition product.

Table 12. User IDs Used for OpenEdition Service

Component Name	Default Service User ID	Base $PPF File Name
CMS	MAINT	2VMVMA10
CP	MAINT	2VMVMB10
AVS	MAINT	2VMVMD10
REXX	MAINT	2VMVMF10
TSAF	MAINT	2VMVMH10
DV	MAINT	2VMVMI10
VMSES	MAINT	2VMVMK10
GCS	MAINT	2VMVML10
SHELL	2VMVMZ10	2VMVMZ10
DCEBASE	2VMVMG10	2VMVMG10
DCEPRIV	2VMVMG10	2VMVMS10
GUI	2VMVMY10	2VMVMY10
LE370	P688198E	5688198E
C390	P654033A	5654033A
ICKDSF	P684042H	5684042H
TCPIP	P735FALF	5735FALF
DIRM	P748XE4M	5748XE4M

Chapter 6. User Administration

This chapter discusses the user administration tasks involved in setting up, configuring, and modifying your system to make users ready for OpenEdition for VM/ESA. After reading this section you should find these activities to be the *enjoyable* part of your administration work. CP directory options, some tools, and detailed information on how to use CSL calls to suit your particular needs will also be discussed.

6.1 The Keys to the UNIX Gate

Root and *superuser* are terms that will not be familiar to most VM/ESA users. These terms come from the UNIX environment. It is important that the VM/ESA system programmer or system administrator know and understand the capabilities of root and superuser.

Root or *superuser* is an authority very similar to the privileges given to the MAINT user ID on VM/ESA. The user owns everything that is needed to build and control the system. With command privilege classes A to G, the user is free to do anything.

In the UNIX environment all I/O is treated as file I/O; for example, hardware devices are represented by a file in directory /dev. Every file is protected by permission settings that govern what a user is allowed to do with that file: read it, write it, or if it is executable, execute it. Root and superuser are not restricted in any way by file permissions, so they can manipulate files as if permissions do not exist. Root is a user given a UID of 0. It is by default the user with the most privilege. Superuser is any user that acquires an effective UID of 0, but did not log in as ROOT.

In the VM/ESA implementation of POSIX, ROOT is also a VM user ID that is enrolled in a BFS file pool. This user ID should be a NOLOG user and should never logged on.

Superuser allows a user to run with an effective UID of 0.

6.1.1 What Is a Root

One definition of “Root” is the beginning of a file system. All files in a file system can be found by tracing a path through the chain of directories starting at the root until the desired file is reached.

In UNIX the phrase “login as ROOT” means to logon to the system using the root ID and root password. The implementation of POSIX in VM/ESA, by default, does not allow a “login as root,” and the root directory is accessible by SFS administrators.

In the VM/ESA POSIX implementation a user would issue the su command to become a superuser, effectively inheriting the root privileges, or “login as root.” Superuser is granted only to users who belong to the system group.

6.1.2 What Is a Superuser

Superuser gives the user complete control over the file system and everything it contains. In a more traditional multiple user UNIX environment, superuser would also give the user control over other users on the system. This is not true for a VM/ESA POSIX environment, where each user is running in a separate machine isolated from other users. The file system may be shared among all users, so any changes made by a superuser to system configuration files will affect other users on the system, but the superuser does not have direct user control. For example, the supervisor cannot see processes that are running in other virtual machines.

System administrators of a VM/ESA system do not usually perform normal user tasks on a user ID with all command classes. It is too easy to make a mistake and erase all system reader files instead of just all reader files for that user, for example, if you are not aware of your current privileges and VM commands.

The same practice should be used when working in the VM/ESA POSIX environment. Do not perform normal tasks as superuser where it is too easy to change something that affects the entire system.

Figure 83 is an example of getting into and out of superuser.

```
openvm shell
IBM
Licensed Material - Property of IBM
5654-030 (C) Copyright IBM Corp. 1995
(C) Copyright Mortice Kern Systems, Inc., 1985, 1993.
(C) Copyright Software Development Group, University of Waterloo, 1989.

All Rights Reserved.
U.S. Government users - RESTRICTED RIGHTS - Use, Duplication, or
Disclosure restricted by GSA-ADP schedule contract with IBM Corp.

IBM is a registered trademark of the IBM Corp.

Thursday April 18 06:58:34 PM EDT 1996
 rodnash /home/rodnash/ $ 1

su
IBM
Licensed Material - Property of IBM
5654-030 (C) Copyright IBM Corp. 1995
(C) Copyright Mortice Kern Systems, Inc., 1985, 1993.
(C) Copyright Software Development Group, University of Waterloo, 1989.

All Rights Reserved.
U.S. Government users - RESTRICTED RIGHTS - Use, Duplication, or
Disclosure restricted by GSA-ADP schedule contract with IBM Corp.

IBM is a registered trademark of the IBM Corp.

Thursday April 18 06:58:39 PM EDT 1996
 rodnash /home/rodnash/ # 2
exit
 rodnash /home/rodnash/ $
exit
Ready; T=0.47/0.59 18:58:56
```

Figure 83. Example of Superuser Mode

In an uncustomized shell, the command prompt changes from a $(1) to a # (2) in indicate superuser mode.

6.1.3 IDs, Real and Effective

A POSIX user ID (UID) is a fullword integer that identifies every OpenEdition VM user ID. It is the sole basis for authority checking against POSIX-defined resources such as the BFS. Each user should be given a unique UID by either the VM/ESA POSIXINFO CP directory statement or by an external security manager product. Any user not explicitly assigned a UID will automatically be assigned a default UID of 4294967295 (X'FFFFFFFF') by DIRECTXA. This is called a user's *real UID*. No user should be permanently given a UID of 0. At times it may be necessary to give a user a UID of 0 to install a product, or when for some reason it is not possible to set an effective UID for that user.

A group ID (GID) is a unique number assigned to a group of related users. This number can be associated with a name, much the same way a UID is associated with a VM user ID.

A user's *effective UID* is that which is allowed by their group membership. The GID is set by either the VM/ESA POSIXGLIST directory statement or by an external security manager product.

The following list summarizes the relationships between UIDs and a GID.

- A user's real UID is defined in the CP directory using a POSIXINFO statement.
- The effective UID is the UID used by the current process.
- The real UID is always equal to the effective UID unless a set-ID file is run.
- You can run a set-ID file if you have CP directory authority to do so.
- When you run a set-ID file, your effective UID becomes the UID of the owner of the file.
- If you belong to a group having GID=0, then you can use the su command, because the file /bin/su has file permissions allowing this.
- By default, su switches your UID to 0.
- su, given the correct arguments, can switch your UID to any other UID.

For more information on UID, refer to 6.3.2, "Adding POSIX Information to the User Directory" on page 145.

6.2 Administering UIDs and GIDs

All OpenEdition for VM/ESA users are members of a *group*. When you give the user a unique POSIX identity by using the POSIXINFO directory statement (see "POSIXINFO Directory Control Statement" on page 146), you also assign which primary group he belongs to. In VM, the CP directory contains this information; in UNIX this information is commonly found in the /etc/passwd file.

You can always tell what your user identity is with the id shell command (not to be confused with the CMS IDENTIFY command). Figure 84 shows the output from these commands (notice the lowercase uid output from the id command).

```
cms id
MINDREAU AT TOTVM1   VIA RSCS     04/22/96 13:35:53 EDT      MONDAY
 mindreau /home/mindreau/ $

id
uid=1102(mindreau) gid=1(staff) groups=4(adm),0(system)
 mindreau /home/mindreau/ $
```

Figure 84. IDENTIFY and id Command Output

In UNIX, groups are used to allow users to share files with other users while excluding other users from the same set of files. If you are familiar with RACF/VM it behaves in much the same way. You grant authorities to a *group* to have them access some data with the same permissions. For example, if you have people in a sales department, you could put all these users in a group called salesgrp. Then they are able to read, write, and execute the same set of files, while people in other departments might not have the same privileges.

Another example is shown in Figure 85. The file posix.terms can be read and written by the owner, read by the staff group, and is inaccessible to the rest of the world.

```
ls -l
total 64
-rw-r-----   1 mindreau staff      22022 Apr 22 14:26 posix.terms
-rw-rw-rw-   1 mindreau system         5 Apr 16 14:19 test.test
-rw-rw-rw-   1 mindreau system       226 Apr  4 15:25 vmgreat.books
 mindreau /home/mindreau/ $
```

Figure 85. GID and UID Permissions to a Particular File

OpenEdition for VM/ESA extends this concept by allowing a user ID to be a member of several groups. The user's supplementary groups may consist of up to 32 different GIDs.

Make sure that in UNIX you define your groups in */etc/group*. In OpenEdition for VM/ESA all group definitions are specified in the CP User Directory.

The choices you make as a system administrator also have a major impact on system security. You have to make the decision to put some people in certain groups on a need-to-know basis according to the kind of work they perform. You have to be very clear in this matter because any security problem that arises will point back to you.

6.3 CP Directory Options and Statements

This section covers the steps involved in assigning POSIX values to users for the implementation of OpenEdition facilities in VM/ESA.

The following topics are discussed:

- First-time handling of the directory source file
- Adding POSIX information to the user directory

POSIX information is divided into a user database and a group database. If an External Security Manager (ESM) is installed, it may be capable of managing these databases and providing the CP with information from them upon request. If not, then the CP obtains this information from the user directory.

6.3.1 First Time Handling of the Directory Source File

Following is a section from the *Program Directory for OpenEdition Shell and Utilities Feature* that discusses the CP user directory and is worth mentioning here:

1 You should place the following definitions in your system's CP directory before building the OpenEdition Shell and Utilities Feature for VM/ESA. You can automate the addition of these definitions by using the DIRPOSIX EXEC on the MAINT 493 minidisk (or 193 minidisk).

Refer to the *VM/ESA: Planning and Administration*, SC24-5750 for more information on DIRPOSIX.

a Add this statement to the MAINT user ID's directory entry *before* any device definitions.

```
POSIXINFO UID 0 GID 0
```

b Place these entries *before* any user statement and common profiles in the CP directory. CP will automatically define the POSIXGROUP DEFAULT, so do not include a statement for this entry.

```
GLOBALDEFS
POSIXGROUP system 0
POSIXGROUP staff 1
POSIXGROUP bin 2
POSIXGROUP sys 3
POSIXGROUP adm 4
POSIXGROUP mail 6
POSIXGROUP security 7
POSIXGROUP nobody 4294967294
```

These definitions (except for the POSIXINFO statement for the MAINT user) are defined for you in the starter system directories. The DIRPOSIX utility discussed next can be used to place them in an already existing directory (for example, in a directory from an established system that has been migrated to a VM/ESA Version 2 Release 1.0 install). The Shell and Utilities feature requires an additional user ID, 2VMVMZ10, to be installed and have a UID and GID of 0.

DIRPOSIX Utility Program

Use the DIRPOSIX utility to add POSIX information to a user directory source file. The normal directory source file in VM/ESA is called USER DIRECT. This directory source can be in either of two forms:

Monolithic format — It consists of a single CMS sequential file that contains all of the virtual machine definitions.

Cluster format — It consists of an index file that points to one or more definition files that contain all of the virtual machine definitions.

DIRPOSIX supports the source directory file in either format, but the file created is always in the monolithic format.

DIRPOSIX carries out the following functions:

- It assigns a unique UID to each user ID that has no *UID specification* and is *not listed* in the DIRPOSIX USEREXCL file.
- It assigns a primary group to each user ID that has *no primary group specification* and is *not listed* in the DIRPOSIX USEREXCL file.
- It adds the standard "system" group definitions and the standard "system" user definitions, if they *do not already* exist.

DIRPOSIX provides a mechanism for reserving installation-specified UIDs; it will not assign any UIDs listed in the DIRPOSIX UIDEXCL file.

DIRPOSIX supports five different options, namely:

MINUID This is the lower bound of the UID range that DIRPOSIX uses when determining the first UID to be assigned. This *minuid* must be between 10 and 4294967000 (X′FFFFFED8′). The default is 200.

MAXUID This is the upper boundary of the UID range that DIRPOSIX uses when determining the UIDs to be assigned. This *maxuid* must be between 10 and 4294967000 (X′FFFFFED8′). Default is 2147483647 (X′7FFFFFFF′).

4294967001 to 4294967293 (4294967295=X′FFFFFFFF′) are reserved for installation use. They will not be assigned by DIRPOSIX.

PRIMarygroup Specifies the primary group to be assigned to each user ID that has no primary group specification and is not listed in the DIRPOSIX USEREXCL file. DIRECTXA and DIRPOSIX support mixed case POSIX group names, so care should be taken when specifying group names.

SYSentries Specifies that DIRPOSIX should add the standard system group definitions and the standard system users.

REPlace Indicates that DIRPOSIX may overwrite an existing output file with the new output file.

Figure 86 shows a sample directory entry for a user after running DIRPOSIX. The command issued was as follows:

```
DIRPOSIX USER DIRECT I (PRIMARYGROUP adm REPLACE
```

```
USER ADMIN01 PWPWPWPW 24M 99M ABCDEG
 POSIXINFO UID 208 GNAME adm
 MACH ESA
 IPL CMS
 CONSOLE  0009 3215
 SPOOL    000C 2540 READER A
 SPOOL    000D 2540 PUNCH A
 SPOOL    000E 1403 A
 LINK MAINT    0190 0190 RR
 LINK MAINT    019E 019E RR
 LINK OPERATOR 0191 0291 RR
 MDISK 0191 3390  0339 0010 210W99 MR
```

Figure 86. DIRPOSIX Result Sample Directory Entry

Some important usage notes:

- DIRPOSIX may be run *multiple* times against the same source directory; this is useful for adding POSIX information to the directory entries of newly added user IDs. DIRPOSIX will assign UIDs beginning one past the largest UID currently in use between available UIDs. You no longer have to worry about duplicated UIDs.

  This information does not apply if you are using an ESM like RACF. If you are using DIRMAINT you may have to run INITLZ to convert the *monolithic* format output of DIRPOSIX into the *cluster* format required by DIRMAINT.
- It is a good idea never to recycle a UID. The range of valid UIDs is sufficient enough so that even a large installation will never need to worry about using the same UID twice.
- Users with no POSIX information in their directory entry have UIDs and primary group specifications assigned by DIRECTXA. A default value of 4294967295 (X′FFFFFFFF′) is assigned if there is no UID specification for a user. If there is no GID or GNAME specification the user′s primary group is the default group, named DEFAULT with GID 4294967295 (X′FFFFFFFF′)
- DIRPOSIX ignores directory PROFILE definitions. They are not assigned UIDs or primary groups by DIRPOSIX, and they are not considered when determining if a user has a UID or primary group specification.
- If SYSENTRIES is in effect, the following POSIX groups are added, if groups with these names are not already defined:

Table 13. POSIX Groups Added by SYSENTRIES

Group name	**GID**
system	0
staff	1
bin	2
sys	3
adm	4
mail	6
security	7
nobody	4294967294

Additionally, the following POSIX users are added, if they are not already defined:

Table 14. POSIX Users Added by SYSENTRIES

User ID	**UID**	**Primary group**
root	0	system
daemon	1	staff
bin	2	bin
sys	3	sys
adm	4	adm
nobody	4294967294	nobody
default	4294967295	DEFAULT

All these added users have a password of NOLOG, indicating that this user is not permitted to log on to the system.

If a special user ID already exists with UID or primary group defined, that information will not be changed. However, the UID or primary group that is not already defined will be updated to the values indicated in the table.

If the UID for a special user ID is already in use, the UID will be assigned to the special user and a warning message will be issued that indicates which other user ID has that UID already assigned to it. This warning is:

Warning, user ID DEFAULT currently has a UID of 4294967294. This is in conflict with a system user. The system user will be added but beware of the ramifications of users sharing UID values.

- The internal form of the SYSAFFIN directory control statement may affect the assignment of a UID or a primary group (or both). If one or more POSIXINFO statements exist for different systems because of SYSAFFIN statements, then a UID and a primary group are not assigned to that user.
- DIRPOSIX LOGFILE is created or appended when DIRPOSIX executes. The logfile is a running history of the changes (informational messages) and of warning messages (potential conflicts) that DIRPOSIX has made. Figure 87 shows a LOGFILE example.

```
------------------------------------------------------------
DirPosix Operation Starting  16 Apr 1996  17:54:27
Command Line Parms: 3391 DIRECT A1 (prim adm
------------------------------------------------------------
Added User    DEFAULT  UID 4294967295   GNAME DEFAULT
Altered User  $ALLOC$  UID        200   GNAME adm
Altered User  $DIRECT$ UID        201   GNAME adm
Altered User  $SYSCKP$ UID        202   GNAME adm
Altered User  $SYSWRM$ UID        203   GNAME adm
Altered User  $PAGE$   UID        204   GNAME adm
Altered User  $SPOOL$  UID        205   GNAME adm
Altered User  $T-DISK$ UID        206   GNAME adm
Altered User  CMS1     UID        207   GNAME adm
Altered User  LGLOPR   UID        208   GNAME adm
Altered User  MAINT    UID        209   GNAME adm
Altered User  CMSBATCH UID        210   GNAME adm
Altered User  AVSVM    UID        211   GNAME adm
Altered User  TSAFVM   UID        212   GNAME adm
Altered User  VMSERVS  UID        213   GNAME adm
Altered User  VMSERVU  UID        214   GNAME adm
Altered User  VMSERVR  UID        215   GNAME adm
Altered User  GCS      UID        216   GNAME adm
Altered User  GCSXA    UID        217   GNAME adm
Altered User  SYSMAINT UID        218   GNAME adm
Altered User  OPERATOR UID        219   GNAME adm
Altered User  OP1      UID        220   GNAME adm
Altered User  OLTSEP   UID        221   GNAME adm
Altered User  EREP     UID        222   GNAME adm
Altered User  OPERATNS UID        223   GNAME adm
Altered User  AUTOLOG1 UID        224   GNAME adm
Altered User  DISKACNT UID        225   GNAME adm
Altered User  OPERSYMP UID        226   GNAME adm
Altered User  VMUTIL   UID        227   GNAME adm
Altered User  SYSDUMP1 UID        228   GNAME adm
Altered User  P684042H UID        229   GNAME adm
Altered User  P688198E UID        230   GNAME adm
Operation complete without severe errors.
```

Figure 87. DIRPOSIX Sample LOGFILE Output

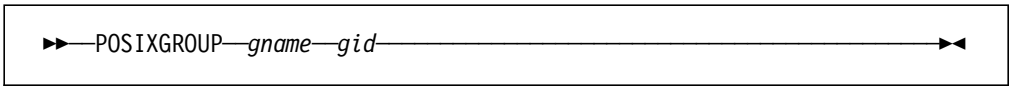
For a complete reference on DIRPOSIX, consult *VM/ESA: Planning and Administration*, SC24-5750.

6.3.2 Adding POSIX Information to the User Directory

There are four POSIX related directory control statements:

1. POSIXGROUP
2. POSIXINFO
3. POSIXGLIST
4. POSIXOPT

POSIXGROUP Directory Control Statement



This statement is used to define your groups to the POSIX database. It must be defined within the global definition section of the source directory (that is, after GLOBALDEFS and before any USER or PROFILE definitions).

GLOBALDEFS is a new directory control statement introduced in VM/ESA 2.1.0. It denotes the start of the global definition section. It is used for SYSAFFIN and GLOBALOPTS. It is where all your POSIXGROUP statements go.

The DIRPOSIX utility inserts the standard group definitions for you after running it for the first time, but you may want to add additional groups following your installation requirements. Figure 88 shows a directory example with POSIXGROUP statements.

```
GLOBALDEFS
 GLOBALOPTS MACHINE XA
 POSIXGROUP system 0
 POSIXGROUP staff 1
 POSIXGROUP bin 2
 POSIXGROUP sys 3
 POSIXGROUP adm 4
 POSIXGROUP mail 6
 POSIXGROUP security 7
 POSIXGROUP nobody 4294967294
 POSIXGROUP general 10
 POSIXGROUP develop 555
```

Figure 88. POSIXGROUP Directory Example

The *gname* operand must be a unique 1- to 8-character mixed-case POSIX group name. The case of this operand is preserved and not converted to uppercase by DIRECTXA. Any group name specified in other directory control statements (like POSIXINFO and POSIXGLIST) that refer to this group must match this group's name exactly.

Note: If you do not want to affect the integrity of the USER DIRECT source file, be sure you have the correct XEDIT CASE setting. You can ensure this by adding SET CASE MIXED to your PROFILE XEDIT file.

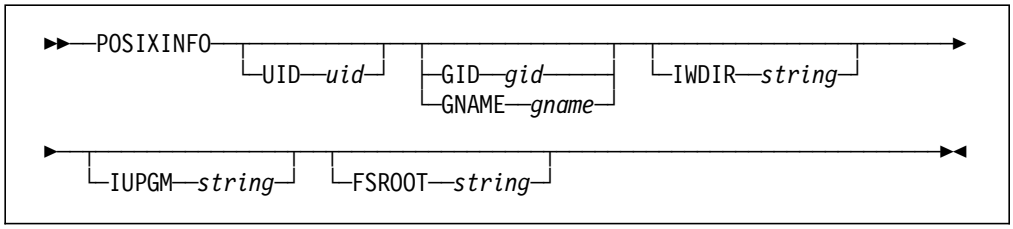
The group name DEFAULT identifies the default group and is reserved by DIRECTXA; it may not be specified on a POSIXGROUP statement. DIRECTXA implicitly defines a default group with a name of DEFAULT and a GID of 4294967295 (X'FFFFFFFF'). This name or GID may be specified on POSIXINFO and POSIXGLIST statements to identify the default group.

The *gid* operand specifies the POSIX group ID (GID) associated with *gname*. The *gid* is a numeric value from 0 to 4294967295 (X'FFFFFFFF').

There is no restriction on the number of POSIXGROUP statements that may be specified, but it is advisable not to define more than you can manage in your directory.

Duplicate group names are not permitted, but multiple groups may have the same GID. This is not recommended, because you may end up with many authorization problems and queries returning values intended for another group.

POSIXINFO Directory Control Statement



This is the most important statement because it defines a user ID's POSIX information in the directory. It contains POSIX *user database* information such as the POSIX user ID, POSIX group ID or group name, initial working directory, initial user program and file system root.

One interesting thing about this statement is that it may be continued across multiple records. Directory control statements continue from one record to the next when the record's last non-blank character in columns 1-71 is a comma (,). The statement is continued beginning in column 1 of the next non-comment, non-blank record. Statement names and keywords may not be split across multiple records.

The following is an example of such a continuation case:

```
POSIXINFO UID 99999 GNAME staff IWDIR '/u/cms1' IUPGM '/bin',
'/sh' FSROOT 'VMBFS:VMSYS:ROOT'
```

Note that you will not get an extra blank between /bin and /sh, because there are certain rules governing quoted string operands. In this case, the result string from *adjacent* quoted strings are concatenated with *no intervening blank*.

UID *uid*

Specifies the POSIX user ID (UID) assigned to this user. It is a numeric identifier. This will be the user's real UID, effective UID and saved set-UID when the user logs on. It identifies the user to the system when certain POSIX functions are being handled, including authorization checks before file access and program execution. Even though multiple users are permitted to have the same UID, this is not recommended because UIDs are used for various authorizations. Also, when a user ID is deleted, the files authorized to that user's UID are left in the file system. These files will be accessible to a new user given the recycled UID. If multiple users have the same UID, individual accountability will be lost.

A UID of 0 identifies a user as one with "appropriate privileges," or superuser. We recommend that *no* user be assigned a UID of 0.

Our recommendation for administration purposes is to make users into members of an authorized group in a need-to-have basis. Then these

special users can intentionally issue the su command. The single exception to this rule are programs that must operate from outside the shell with ROOT privileges. Often, this is needed during product installation.

The *uid* is a numeric value between 0 and 4294967295 (X′FFFFFFFF′). A default value of 4294967295 (X′FFFFFFFF′) is assigned if there is no UID specification for a user. This is the only acceptable case where users can obtain the same UID.

GID *gid*, **GNAME** *gname*
This specifies the user′s primary group. The *gid* is a numeric value between 0 and 4294967295 (X′FFFFFFFF′). The *gname* is a 1- to 8-character mixed-case POSIX group name. The case of the group name is preserved (the same rules apply as for POSIXGROUP). The *gid* or *gname* must identify a group defined on a POSIXGROUP directory control statement. The GID for this group will be the user′s real GID, effective GID and saved-set GID when the user logs on. A user′s group affiliation is referenced when certain POSIX functions are being handled, including authorization checks before file access and program execution.

If users are not assigned a GID in the user database, they may each be assigned the same default value by the system. In this case, all of these users will appear the same to POSIX functions that reference these GIDs. This could be used to permit all of these users to access or run certain files, or it could be used to deny all unregistered users from accessing or running any POSIX file.

IWDIR *string*
This specifies the user′s initial working directory, or home directory. Unless overridden by CMS′s OPENVM SET DIRECTORY command, it will be the current directory when a user first enters the POSIX environment. If this value does not point to an existing directory, unexplained errors could happen when the user enters the shell.

IUPGM *string*
This specifies the user′s initial user program. It is the name of an application. It is typically a shell which is a program that accepts commands from the user and supervises the execution of other programs. This is the program that will be invoked by the OPENVM SHELL command. In most cases, there is no need to specify an initial user program.

FSROOT *string*
This specifies the user′s file system root. Though not part of POSIX requirements, it is contained in the user database. Unless overridden by CMS′s OPENVM MOUNT command, the Byte File System will be mounted as the root file system when a user first enters the POSIX environment.

The *string* specifying an IWDIR, IUPGM or FSROOT results in a mixed-case 1- to 1023-character string which may contain blanks, single/double quotation marks and other special characters. The *string* begins with the first non-blank character following its keyword and may be continued on multiple directory records.

For more information on “Quoted String Operands” see *VM/ESA: Planning and Administration*, SC24-5750.

POSIXINFO statements are permitted in profile entries, but specifying the UID in a profile entry is not recommended because it might result in many users having the same UID.

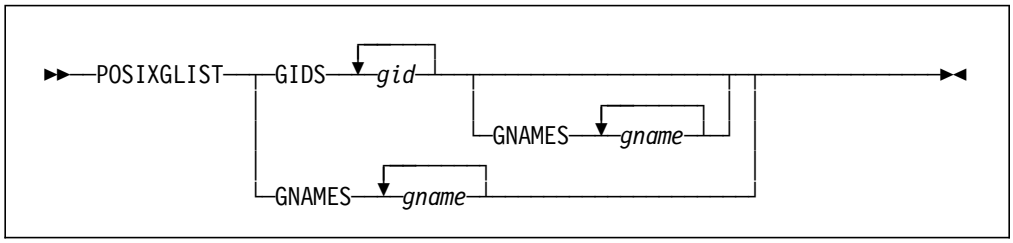
POSIX *user names*, sometimes called *login names*, are defined to be the *lowercase* version of the user's VM user ID.

Figure 89 shows an example of POSIXINFO.

```
POSIXGLIST GIDS  4 GNAMES  system
POSIXINFO UID 1102 GNAME staff IWDIR '/u/mindreau' IUPGM '/',
 '/bin/sh' FSROOT 'VMBFS:VMSYS:ROOT'
POSIXOPT SETIDS ALLOW EXEC_SETIDS ALLOW
```

Figure 89. POSIXINFO Directory Example

POSIXGLIST Directory Control Statement



This statement is used to list POSIX groups of which the user is a member. POSIXGLIST statements are the primary source for the GIDs that form the user's supplementary group list.

If you specify the POSIXGLIST statement, it must precede any device statements you specify in a profile or user entry.

GIDS *gid*
: Each *gid* must identify a group defined on a POSIXGROUP statement or the default group implicitly defined by DIRECTXA.

GNAMES *gname*
: Specifies POSIX groups of which the user is a member. Each *gname* must identify a group defined on a POSIXGROUP statement or the default group implicitly defined by DIRECTXA. The case of each group name is preserved; it is not converted to upper case by DIRECTXA. Each name must exactly match the group name on a POSIXGROUP statement or the default group name.

Duplicate *gids* or *gnames* are not allowed. Applications using database queries such as `getgrnam()` and `getgrgid()` that require a GID as input may return information about a group other than the intended one if the same GID was assigned.

The user's primary group defined by a POSIXINFO statement may be listed on a POSIXGLIST statement, but it is not required to be listed. A user is automatically considered a member of his primary group.

POSIX has the concept of *supplementary groups*. Some of the groups of which a user is a member are defined to that user's supplementary groups. This list of groups is referenced when certain POSIX functions are being handled, including authorization checks before file access and program execution. The user's supplementary GID list consists of up to 32 unique GIDs. The primary GID (from

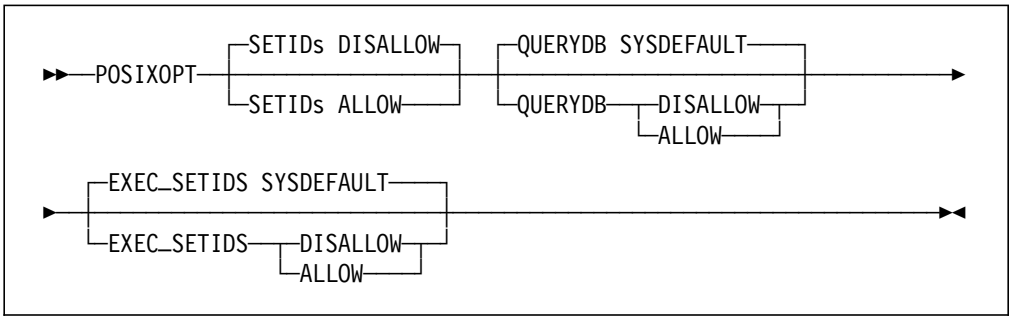
the POSIXINFO statement) is always included in the initial list. The remainder are taken from the POSIXGLIST statement in the order in which they were specified.

Figure 90 shows an example of POSIXGLIST.

```
POSIXGLIST GIDS  4 GNAMES  system
POSIXINFO UID 1102 GNAME staff IWDIR '/u/mindreau' IUPGM '/',
'bin/sh' FSROOT 'VMBFS:VMSYS:ROOT'
POSIXOPT SETIDS ALLOW EXEC_SETIDS ALLOW
```

Figure 90. POSIXGLIST Directory Example

POSIXOPT Directory Control Statement



This statement specifies option settings related to a user ID's POSIX capabilities. It includes authorizations to query and set POSIX process and database information.

This statement will be discussed in more detail in 7.4.2, "POSIXOPT QUERYDB" on page 164.

Figure 91 shows an example of POSIXOPT.

```
POSIXGLIST GIDS  4 GNAMES  system
POSIXINFO UID 1102 GNAME staff IWDIR '/u/mindreau' IUPGM '/',
'bin/sh' FSROOT 'VMBFS:VMSYS:ROOT'
POSIXOPT SETIDS ALLOW EXEC_SETIDS ALLOW
```

Figure 91. POSIXOPT Directory Example

6.4 CSL Calls Specific to UIDs and GIDs

The CMS callable services library (CSL) is a facility that lets application programs written in REXX or other high-level languages access OpenEdition for VM/ESA services. The CSL makes application programming easier because you do not need to program complex assembler routines.

OpenEdition for VM/ESA callable services reside in VMMTLIB, which is located in the CMS nucleus. These CSL routines provide functions for manipulating BFS files and data.

The following callable services are specific to POSIX GIDs and UIDs:

BPX1GGI Access the group database by ID

BPX1GGN Access the group database by name

BPX1GPU Access the user database by user ID

BPX1GPN Access the user database by user name

BPX1GEG Get the effective group ID

BPX1GID Get the real group ID

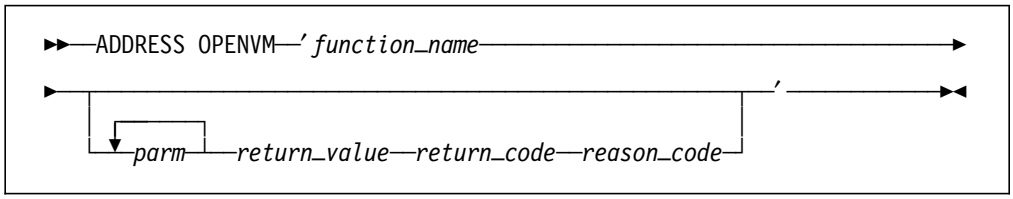
BPX1GEU Get the effective user ID

BPX1GUI Get the real user ID

BPX1GGR Get a list of supplementary group IDs

BPX1GUG Get a list of supplementary group IDs by user name

Here is the format to use when calling an OPENVM routine from a REXX program:



function_name
is the name of the OPENVM routine to be called.

parm
is the name of one or more parameters to be passed to the OPENVM routine. The number and type of these parameters are routine-dependent. A parameter being passed must be the name of a variable.

One of the following conditions arises after calling an OPENVM routine:

1. If REXX successfully calls the OPENVM routine, then the REXX variable *rc* contains a zero return code. The values of *return_value, return_code,* and *reason_code* in the parameter list are set by the routine. For more information on these values see *IBM OpenEdition for VM/ESA: Callable Services Reference*, SC24-5726.
2. If REXX detects an error and is *not* able to call the OPENVM routine, then the REXX variable *rc* contains a *negative* return code. See “Return Codes” under “Invoking OPENVM Routines” in *VM/ESA: REXX/VM Reference*, SC24-5770.
3. If your *rc* variable returns a *-3*, then you may not be running on VM/ESA Version 2 Release 1.0. This product is a prerequisite to run OPENVM routines.

The following parameters are explained in more detail:

Return_value This parameter indicates the success or failure of the service. If the callable service fails, it returns a -1. For most successful calls to OpenEdition services, the return value comes back a 0.

However, some services, such as BPX1GGI and BPX1GGN, return zeros instead of -1 when the service fails.

Note: This is not the case for services called from a REXX EXEC. In REXX you should check the *rc* variable.

Return_code This parameter is referred as the *errno* in the POSIX C interface. The *return_code* is returned only if the service fails.

Reason_code This parameter usually accompanies the *return_code* value when the callable service fails. It further defines the return code. The reason codes are equivalent to the C errno2 numbers.

All *return_codes* and *reason_codes* are listed in Appendix A and B in *IBM OpenEdition for VM/ESA: Callable Services Reference*, SC24-5726.

6.4.1 CSLPSCAN, POSIX Option Reporting Tool

CSLPSCAN creates a report on the POSIX options defined in the CP directory. It uses many of the CSL routines discussed in this section. The program source for CSLPSCAN and a complete discussion of the program logic can be found in section B.1, "CSLPSCAN CP Directory POSIX Option Reporting Tool" on page 183. Figure 92 shows the result of running CSLPSCAN. CSLPSCAN accepts a group name or GID as an argument, then locates information about users through the CSL calls. Access to the CP directory source is not required.

```
cslpscan system
CSLPSCAN: ____________________________________________________________
CSLPSCAN: Information for group system(0)  Total number of members: 16.
CSLPSCAN: ____________________________________________________________
CSLPSCAN: Username        UID        GID IWDIR    IUPGM
CSLPSCAN: -------- ---------- ---------- -------- --------------------
CSLPSCAN: cms1          99999          1 /bin/sh  /u/cms1
CSLPSCAN: dceadmin          0          0 /bin/sh  /
CSLPSCAN: dcecore           0          0 /bin/sh  /
CSLPSCAN: dcevmbfs          0          0 /bin/sh  /
CSLPSCAN: dfsms        123321          1 /bin/sh  /u/maintmo
CSLPSCAN: jens          73645          1 /bin/sh  /u/jens
CSLPSCAN: jens2         73646          1 /bin/sh  /u/jens2
CSLPSCAN: maint            10          1 /bin/sh  /u/maint
CSLPSCAN: maintmo       73226          1 /bin/sh  /u/maintmo
CSLPSCAN: maintrt       12623          1 /bin/sh  /u/maintrt
CSLPSCAN: maintsv       47360          1 /bin/sh  /u/maintsv
CSLPSCAN: mindreau       1102          1 /bin/sh  /u/mindreau
CSLPSCAN: rodnash        1002          1 /bin/sh  /u/rodnash
CSLPSCAN: root              0          0 /bin/sh  /
CSLPSCAN: 2vmvmg10          0          0 /bin/sh  /
CSLPSCAN: 2vmvmz10          0          0 /bin/sh  /
CSLPSCAN: ____________________________________________________________
Ready; T=0.03/0.04 13:40:13
```

Figure 92. CSLPSCAN EXEC Sample Output

6.4.2 DIRPSCAN, Directory Reporting Tool

The DIRPSCAN EXEC reads the CP directory in monolithic format as an argument. The default input is USER BACKUP *. It searches the directory for POSIX options, then creates a report. The complete source of DIRPSCAN can be found in section B.2, “DIRPSCAN, a Directory Reporting Tool” on page 189.

Figure 93 shows the output of DIRPSCAN. The character A shown under the SQE heading indicates at least one of the following attributes are set:

A-- User has POSIXOPT SETIDS ALLOW
-A- User has POSIXOPT QUERYDB ALLOW
--A User has POSIXOPT EXEC_SETIDS ALLOW

```
dirpscan
DIRPSCAN: ------------------------------------------------------------
DIRPSCAN: POSIXGROUP  GroupID (GID)   Hex GID        Gaps (if any)
DIRPSCAN: ------------------------------------------------------------
DIRPSCAN: system               0      X'00000000'
DIRPSCAN: staff                1      X'00000001'
DIRPSCAN: bin                  2      X'00000002'
DIRPSCAN: sys                  3      X'00000003'
DIRPSCAN: adm                  4      X'00000004'
DIRPSCAN:                                            5-5
DIRPSCAN: mail                 6      X'00000006'
DIRPSCAN: security             7      X'00000007'
DIRPSCAN:                                            8-9
DIRPSCAN: general             10      X'0000000A'
DIRPSCAN:                                            11-554
DIRPSCAN: develop            555      X'0000022B'
DIRPSCAN:                                            556-4294967293
DIRPSCAN: nobody       4294967294     X'FFFFFFFE'
DIRPSCAN: default      4294967295     X'FFFFFFFF'
DIRPSCAN: ------------------------------------------------------------
DIRPSCAN: POSIXINFO       UID GNAME    SQE POSIXGLIST
DIRPSCAN: ------------------------------------------------------------
DIRPSCAN: BFS                          A--
DIRPSCAN: BFSERVU                      A--
DIRPSCAN: VMSERBFS                     A--
DIRPSCAN: VMSERVS                      A--
DIRPSCAN: VMSERVU                      A--
DIRPSCAN: ROOT               0 system
DIRPSCAN: DAEMON             1 staff
DIRPSCAN: BIN                2 bin
DIRPSCAN: SYS                3 sys
DIRPSCAN: ADM                4 adm
DIRPSCAN: MAINT             10 staff    A-A system
DIRPSCAN: CMS2             666          A-A develop
DIRPSCAN: DCE11           1001          A-A staff
DIRPSCAN: TIBO           12624 staff    AAA
DIRPSCAN: MAINTAV        43018 staff    A-A
DIRPSCAN: MAINTSV        47360 staff    A-A system
DIRPSCAN: MAINTMO        73226 staff    A-A system
DIRPSCAN: JENS           73645 staff    A-A system
DIRPSCAN: CMS1           99999 staff    A-A system
DIRPSCAN: DFSMS         123321 staff    A-A system
...
```

Figure 93. DIRPSCAN EXEC Sample Output

6.5 Creating a Home Directory

To create a repository for the shell history file, files from the `mailx` utility, profiles, and other data, all POSIX users should have a *home directory*. To create the home directory for user `kristien`, an administrator may use the following command sequence in the shell:

```
cd /home
umask g=rx,o=rx
mkdir kristien
chown kristien:dept75 kristien
cd kristien
cp /etc/.profile.sample .profile
chmod u=rw,g=r,o= .profile
chown kristien:dept75 .profile
```

The `mkdir` command creates the directory and `chown` gives ownership to `kristien` and `dept75`. A sample .profile is copied, its file permissions set, and then ownership of the file is given to `kristien`. She is now able to modify her `.profile` to suit her personal needs, and she has an area on the file system she can call her own.

Figure 94 shows an example shell script used as a general facility to create a user's home directory.

```
# ------------------------------------------------------------------------
# Create a Home Directory for a New User
# ------------------------------------------------------------------------
userid=$(id -u)                                  # get our userid
id $1 > /dev/null                                # see if user exists
if   test $? -ne 0                               # user exists?
 then echo 'adduser: userid "'$1'" is invalid'   # invalid userid...
elif test $# -eq 0                               # not fooled by our own
 then echo 'adduser: Please enter a userid'      # we need a userid
elif test $userid -ne 0                          # check if superuser
 then echo 'adduser: Sorry, to run this cmd first "su" (if authorized)'
      else echo 'adduser: Beginning process for user' $1'...'
           cd /home                                                  1
           umask g=rx,o=rx
           mkdir $1                                                  2
           chown $1:${2:=staff} $1                                   3
           cd $1                                                     4
           cp /etc/.profile.sample .profile                          5
           chown $1:$2 .profile                                      6
           chmod u=rw,g=r,o= .profile                                7
           echo 'adduser: Finished process.  This is the result:'
           ls -dl /home/$1
           ls -al /home/$1
fi
```

Figure 94. ADDUSER Shell Script

The same operations can be accomplished using OPENVM commands, however they must be run from a VM user ID with a UID of 0, file server administor's authority, or proper BFS group memberships. Figure 95 on page 154 shows an example REXX EXEC used to create a user's home directories.

```
/*---------------------------------------------------------
   Create a Home Directory for a New User
---------------------------------------------------------*/
trace 'O'
parse arg userid group
if userid = '' then do
  say 'You must enter a valid user ID'
  call exit_it
end
if group = '' then do
  say 'Using default group staff'
  group = 'staff'
end
'OPENVM SET MASK rwx r-x r-x'
'OPENVM CREATE DIRECTORY /home/'userid                          2
'OPENVM OWNER /home/'userid group userid                        3
'OPENVM GETBFS /etc/profile.sample /home/'userid'/.profile'     5
'OPENVM OWNER /home/'userid'/.profile' group userid             6
'OPENVM PERMIT /home/'userid'/.profile rw- r-- ---'             7
'OPENVM LIST /home/'userid '(OWNERS OBJECT'
'OPENVM LIST /home/'userid'/.profile (OWNERS'
exit_it:
  exit
return
```

Figure 95. ADDUSER EXEC

1 Move to the standard directory where user data is kept.

2 This is the command that creates the directory.

3 Change the owner of this directory to the user in question.

4 Move to the newly created directory.

5 Copy a sample profile to assist the new user.

6 Change this `.profile` ownership to the new user ID.

7 Set the permission bits to recommended security settings.

6.6 Logging, Messaging, and Broadcasting

This section briefly discusses logging facilities in OpenEdition for VM/ESA and the ability to send messages to other people in your system or other systems.

6.6.1 Logging Facilities

Logging is the facility a system exploits to record system-related activity about specific events. You can also record user-related activities originated by a user. Most of the time a *log* is used to aid in the investigation of specific events that lead to an error situation.

Log Files

All commands that are input during an interactive shell session are stored in a history file. By default the history file is `$HOME/.sh_history`, and it stores, by default, the last 128 commands entered. Figure 96 on page 155 shows what a `.sh_history` file looks like.

```
cat .sh_history
  cms 'spool console start * class a'
  ls -la
  cms 'spool console close'
  cms 'receive 21 sample console a'
  cat //sample.console.a
  cms rl
  cat .sh_history
mindreau /home/mindreau/ $
```

Figure 96. .sh_history Contents Sample

Most UNIX implementations supply a facility to save messages to a log file. This is controlled by a system message facility, *syslogd* daemon, and related utilities. This daemon is not part of the POSIX standard and not implemented in OpenEdition for VM/ESA.

VM/ESA Log Files

Logging of user-related activities is accomplished by spooling your virtual console. The CP SPOOL command provides the CONSOLE operand, which lets you begin and end console spooling. You can enter:

```
spool console start
```

when you want to begin recording your terminal session, and enter:

```
spool console stop
```

when you have finished. Between, you can periodically close the console file to release for printing whatever has been spooled thus far, by entering:

```
spool console close
```

You can even drop it if you do not wish to keep it, by entering:

```
spool console purge
```

An alternate technique is to spool your console to your own virtual reader, by entering:

```
spool console start * class a
```

Then, when you close the console file, instead of being released to the CP printer spool file queue, it is routed to your virtual reader and you can load it onto an accessed minidisk or SFS directory as a CMS file. Figure 97 on page 156 shows an example of how to create a file from your console records. You can then use the XEDIT editor to examine it (or delete sections you do not need) and use the PRINT command (or a similar installation-dependent facility) to create a printed copy.

```
cms 'spool console start * class a'
 mindreau /home/mindreau/ $
ls -la
total 72
-rw-------   1 mindreau system       373 Apr 23 14:09 .sh_history
-rw-r-----   1 mindreau staff      22022 Apr 22 14:26 posix.terms
-rw-rw-rw-   1 mindreau system         5 Apr 16 14:19 test.test
-rw-rw-rw-   1 mindreau system       226 Apr  4 15:25 vmgreat.books
 mindreau /home/mindreau/ $
cms 'spool console close'
RDR FILE 0021 SENT FROM MINDREAU CON WAS 0021 RECS 0009 CPY  001 A
 mindreau /home/mindreau/ $
cms 'receive 21 sample console a'
DMSRDC738I Record length is 132 bytes
SAMPLE CONSOLE A1 created
File SAMPLE CONSOLE A1 received from MINDREAU at TOTVM1 sent as (none)
 mindreau /home/mindreau/ $
cat //sample.console.a
 mindreau /home/mindreau/ $
ls -la
total 72
-rw-------   1 mindreau system       373 Apr 23 14:09 .sh_history
-rw-r-----   1 mindreau staff      22022 Apr 22 14:26 posix.terms
-rw-rw-rw-   1 mindreau system         5 Apr 16 14:19 test.test
-rw-rw-rw-   1 mindreau system       226 Apr  4 15:25 vmgreat.books
 mindreau /home/mindreau/ $
cms 'spool console stop'
 mindreau /home/mindreau/ $
```

Figure 97. Creating a Console Output File

Notice that in the shell you should not enter the cp command to communicate to the Control Program like you do in CMS. In the shell, cp is the *copy file* command. You need to use the shell cms shell command to issue Control Program commands. You may have noticed the quotes surrounding the CMS command parameters. This is because CMS command syntax includes the characters "*) and (." These characters must be enclosed in single or double quotes to prevent the shell from interpreting them.

Even if you are disconnected from VM/ESA, all output generated by your virtual machine will continue to be spooled.

In VM/ESA the logging of system-related activities is just the same. User ID OPERATOR (or another user ID designated as the system operator) is used here. This user ID is automatically logged on to the system every time VM/ESA is initialized and comes up with a virtual console started. This user ID is *special* because all system-related messages are written to its console.

Many licensed programs route messages to this console too, and hence you obtain a complete image of system activities. At times, the number of messages is too overwhelming and a *Programmable Operator* is run to filter out the most important messages and route them to a *logical operator* for appropriate actions.

6.6.2 Messages and Broadcasting

In UNIX, the logger utility can be used to send messages to the *syslog* facility. In OpenEdition for VM/ESA, the logger command sends messages by default to the operator's console. Also, logger allows you to specify a destination where you can send a message even if the user is on a remote system. CMS uses the TELL command to transmit your log message.

Figure 98 shows how to use the *logger* command and how the output appears on the receiver's side.

```
* this is user TOTVM1(MINDREAU)
logger -T -d 'mindreau at wtscpok' VM/ESA is by far the most
brilliant way to run your business
 mindreau /home/mindreau/ $

* this is user WTSCPOK(MINDREAU)
 From TOTVM1(MINDREAU): 04/23/96 17:15:51: mindreau: 3957: VM/ESA is by
  far the most brilliant way to run your business
```

Figure 98. .sh_history Contents Sample

The **-**d (destination) parameter treats arguments in a way that is similar to the CMS TELL command. You can use *nicknames* to send messages to a particular user or list of users. If you are authorized, you may send a message to *all*, effectively sending a broadcast type of message.

6.7 Deleting a POSIX User

Deleting the POSIX definitions from a user ID's CP directory may have several side effects. You must chown all the objects that belonged to the user in the Byte File System. As superuser, the following command changes the ownership of all the objects from maintsv to the user bfsmaint. This includes files and directories.

```
find / -user maintsv -exec chown bfsmaint {} \;
```

It is a good idea to have a user ID, such as BFSMAINT, become the owner of any orphaned files. This will prevent a new user that is assigned a recycled UID from obtaining access to old undeleted files. It is even a better idea never to recycle a UID.

When deleting POSIX definitions, you must search every file system that exists on your VM system. VM/ESA does not limit the number of Byte File Systems.

If a file is orphaned, or the UID of the file owner no longer exists in the CP directory, the UID number will replace the owner field when the file is listed with the ls shell command. The normal owner displayed is the CP user ID name.

Chapter 7. Security

This chapter introduces you to several UNIX security topics that apply to the POSIX implementation in OpenEdition for VM/ESA.

7.1 Some Basic Concepts

Security and its implementation have been discussed in many documents. The full discussion will not be repeated here. However, because of the critical importance of this topic, you will now be presented with a short review of the more important subjects.

7.1.1 Security Policies

Security policy considerations should include:

- The security boundaries between individuals, groups, the organization, and the rest of the world. What types of data must be restricted to specific individuals? What is shared by departments or other groups? What is available to everyone in the organization? What can be freely distributed outside the organization?
- How should the organization be divided into groups (for security purposes)? This may not always follow departmental lines. (In practice, considerable effort is needed to produce good group definitions.)
- What levels of security are needed for these various functions? It is important not to overstate the needs. The requirements of a clearing bank differ from those of a clothing manufacturer. Any determination should include rough classifications based on confidentiality (controlled read) and integrity (controlled write).
- Where is the line between nice-to-have and required security? Specifying too much security (to cover all the nice-to-have situations) can detract from required security. The trivial statement “All our company information should be secure” is not a reasonable starting (or ending) point for a security policy.
- What is the cost of security? What is an acceptable cost? Administration and inconvenience to users are parts of the total cost. What is the risk and cost of lost or damaged or compromised data? Often a reasonable risk/cost analysis (using high/medium/low categories rather than monetary costs) can eliminate many of the nice-to-have items from a security-controls list.
- Who administers security? Who monitors or audits it? These functions require time and skill. *No amount of policy tuning or management directives will make security take care of itself*.
- What is the importance of the security policy to the individual and the organization? What are the enforcement mechanisms? For example, in some organizations a person might be casually reminded about security violations. In other organizations the same violation might result in immediate dismissal.

7.1.2 Who Needs Security?

Security often invokes images of industrial espionage and hackers. This is a very small element of security. The major reasons for computer security include:

1. Reduction of errors. A keying error may delete the wrong file, for example. Secured files and a good backup procedure will reduce these problems.
2. Disgruntled employees (or former employees). These may be after confidential information or may wish to sabotage the system.
3. Curious employees. For example, everyone would like to read (and perhaps change) payroll and personnel files.
4. Thrill-seeking employees. Breaking a security system is a diverting exercise for some people. No harm is intended. However, if confidential information is to remain confidential, it must be guarded.
5. There may be legal considerations involved with system security. If due professional care is not exercised, an organization might be liable for losses caused by poor computer security.

Several countries have laws about databases and security. Your system, as a whole, might be considered a database. These laws may cover anything that might be considered ledger data, information that might influence stock prices, personnel information, and so forth.

Security must be reasonably consistent across an organization. There is no point in securing a file on one system when a copy of the file can be openly read on another system. Again, security begins with top management and the policy (goals) should be understood throughout the organization.

7.1.3 How Much Security?

Too much security may be as bad as too little security. Too much security, if poorly implemented and administered, may be ineffective and make a system very difficult to use. Increasing system security requires increasing administrative time and effort.

Computer security levels defined by the U.S. Department of Defense have become a point of reference for purchasing systems. These levels, in increasing order, are D, C1, C2, B1, B2, B3, and A.

- Level **D** has no certified security functions.
- The **C** levels have discretionary security controls. That is, the user decides which resources to protect and controls (to some extent) how the protection is applied.
- The **B** levels have mandatory security controls (along with other additional functions). The controls are automatically applied by the system.
- The **A** level is very unusual and only a few examples exist. (These levels are in the context of isolated systems and generally ignore networking issues.)

Installing a B-level system does not automatically answer security concerns. Considerably more planning and administrative time may be needed to administer a B system than a lower-level system. A system certified for B-level probably will not run at this level when using only the default security options. Continuing administrative effort is needed to establish and maintain the B level

functions. Do not purchase or install more security than you plan to administer. Administration is not free, and it is not automatic.

Note: Except for a few items, most of the C2 and B1 criteria are already built into the VM/ESA system. You can use RACF Version 1 (5740-XXH) Release 9.2 (or its equivalent) if a full security product is required. Release 1.9.2 for VM, in combination with a specific set of VM/ESA products, provides support designed to meet the United States Department of Defense B1 security criteria. For more information on this subject see *VM/ESA: C2/B1 Trusted Facility for VM/ESA with RACF*, SC24-5563.

7.2 General Security Structure

OpenEdition for VM/ESA shares many of the same general security structures as UNIX. The security mechanisms are integral to the VM/ESA operating system. Following is a discussion of the key elements of UNIX and how OpenEdition for VM/ESA interacts with them.

Supervisor State and Problem State Operation: All modern processors have at least two hardware-enforced states of operation. A program operating in *supervisor state* can perform any CPU instruction, alter any register, and so forth. A program operating in *problem state* can use a limited set of instructions and cannot access some of the control registers of the processor. Programs operating in supervisor state can bypass any hardware-provided controls, while programs in problem state are bounded by restrictions and the environment set by the operating system.

In VM/ESA only the CP component is allowed to run in supervisor state. All virtual machines run in problem state (not taking into account the Start Interpretive Execution (SIE) assist environment).

Kernel Mode and User Mode: The terms supervisor and problem states are not extended to the UNIX programming interface. UNIX does provide the concept of *kernel mode* and *user mode* levels, and these correspond to supervisor and problem states. You can add additional programs to your UNIX kernel. UNIX makes this easy by using dynamic linking to load kernel modules.

You can think of the kernel in VM/ESA as a collection of programming interfaces implemented by several VM/ESA components such as CP and CMS (including CSL, VMMTLIB and others). There is no way a general POSIX user (even a superuser) could change this interface without the proper authorizations.

Programs Operating with Root Authority: Under UNIX, programs operating with root authority automatically skip many security checks. Ignoring file permission bits is the most obvious omission. This operation of root is part of the defined UNIX interface. Programs operating as root are not the same as kernel-mode programs. Except for special cases in the kernel, root programs are in user mode (which is in CPU problem state). Programs operate with root authority because they are called by another process that is operating with root authority, or because they set their UID to root.

The *superuser* is a privileged account associated with the user name *root*. The su command is a privileged command in OpenEdition for VM/ESA.

File and Directory Authorization: All UNIX systems have permission bits to control access to files and directories. UNIX extends the notion of files to practically everything in the system. Devices are files; system memory itself can be regarded as a file. Permission bits can protect all these resources.

Support for secure file access is also provided in OpenEdition for VM/ESA. Authorization for file access is validated by checking the POSIX security values of the requestor against those permitted to access the file. An external security interface is provided for those wishing to have an external security manager (ESM). An example of this ESM is RACF/VM. It provides the POSIX group and user database information and authorizes access to the POSIX resources. The ESM can also authorize changes to POSIX processes' IDs. For more information on permission, see 2.4, "Understanding File System Permissions" on page 11.

It should be noted that a Shared File System administrator has access to all data within it. People not familiar with BFS security should not be SFS administrators.

Login Authentication: The login process establishes the user's identity. Several programs perform this function, such as login, TELNET, FTP, and so forth. These programs use root authority to establish the new user's identity. The login function is not always used. Any root program can assume the identity of any defined user, effectively skipping the login process for that user.

In OpenEdition for VM/ESA, CP controls the LOGON process. You have no access to the system when you do not provide VM with a correct *user ID* and *password* combination. Your CP login becomes your POSIX login.

7.3 Reviewing Your System

The following considerations and tests may be useful for reviewing the security environment of your POSIX system.

- Are groups used effectively? Job assignments should be divided and grouped based on potential security exposures.
- Is there a written security policy (no matter how simple it may be)? Policies for passwords, unattended terminals, and restricted documents, to name a few, should be well understood by all employees.
- Who takes backups? This question is less important for a big installation (where a complete system backup is well planned) than for small sites where a backup strategy for the file pool system should be implemented. Do several people know where backup records are kept? Can several people find the most recent backup and restore files from it?
- The default permissions for a $HOME directory are 755 or `rwxr-xr-x` (for an explanation on this octal code, see Table 2 on page 12). This allows anyone to display a user's $HOME directory; this may be appropriate for your installation. Use the command `ls -l /home` to display the permissions for all users $HOME directories (assuming you are using the normal UNIX directory structure). HOME directory permissions 710 (`rwx--x---`) prevents anyone except the owner of the directory from writing in it. It also prevents groups and others from visiting a user's HOME directory, and searching for files with insecure permissions or interesting information. A permission mode of 711 (`rwx--x--x`) on a user's HOME directory allows others to traverse to lower directories within the HOME directory which have more open permissions,

but not to display the contents of the base $HOME directory. Files that require access by others should be placed in lower directories.

- Initialization files, such as $HOME/.profile, should not be writable by anyone other than the owner. The permission modes for these files should be 640 or 600 (`rw-r-----` or `rw-------`). If writable by others, they could be modified in a way to compromise all files to which the user has write access, or the user's PATH could be changed. This is a critical check. Try the command in Figure 99.

```
find /home -name .profile -exec ls -l {} \;
-rw-rw-rw-   1 jens     system       217 Oct  3  1995 /home/jens/.profile
-rw-rw-rw-   1 jens2    staff       1010 Sep 28  1995 /home/jens2/.profile
-rw-rw-rw-   1 dceadmin system       415 Sep 28  1995 /home/maintav/.profile
-rw-rw-rw-   1 dceadmin system       407 Sep 15  1995 /home/maintmo/.profile
-rwxr-----   1 maintrt  system      1306 Sep 25  1995 /home/maintrt/.profile
-rw-rw-rw-   1 dceadmin system       523 Apr 10 10:00 /home/rodnash/.profile
-rwxr-xr-x   1 tibo     staff        256 Sep 26  1995 /home/tibo/.profile
 mindreau /home/mindreau/ $
```

Figure 99. Finding Permissions for a Particular File

- The default permission mode (a user's umask) should be set to 022 or 027. The default umask can be set in `/etc/profile`.
- The user's path should be reviewed to ensure that system directories are checked before local directories and the current working directory is not defined in PATH.
- Review hidden files. Some users think they can hide various misdeeds in them. You may also uncover large amounts of wasted disk space. The command `find / -name .??* -exec ls -l {} \;` can be used to view hidden files. Redirect the output to a file, since there may be a large output.
- System directories should have permission mode at least as restrictive as 755 (`rwxr-xr-x`) Examine all *world-writable* directories. A sample command is `find / -perm -002 -type d -exec ls -ld {} \;`.

  World-writable directories owned by root or bin should be examined with some care. (You should have root privileges to use the above command to avoid generating any error messages).
- User home directories that are world-writable are common and are exposures. You can find these with: `find /home -perm -002 -type d -exec ls -ld {} \;`.

  There are few excuses for a user's home directory to be world writable. Attacks through such directories are a basic tool of system intruders.
- All device files should reside in `/dev`. Any device files residing outside `/dev` should be investigated.
- Check for unowned files. The command `find / -nouser -print` can be used.

7.4 VM Extended Security Controls

OpenEdition for VM/ESA allows for the extension to the access security mechanisms by providing a program interface.

We recommend the use of RACF or another ESM to provide an audit trail of access violations and attempts. Another recommendation is to never globally define an authorization, such as EXEC_SETIDS on the POSIXOPT directory statement. It is better to authorize all the active POSIX users on an individual basis than to give authority to every user on a system.

7.4.1 RACF Version 1 Release 10 for VM

RACF (program number 5740-XXH), an External Security Manager (ESM), provides support for OpenEdition for VM/ESA which contributes to the strength of System/390 open client/server environment and aids in application-level portability on VM/ESA. It provides the capability to register OpenEdition for VM/ESA users and groups in the RACF database and offers security for files and directories residing in the OpenEdition Byte File System.

With RACF, security is provided at the POSIX (Portable Operating System Interface) 1003.1 level:

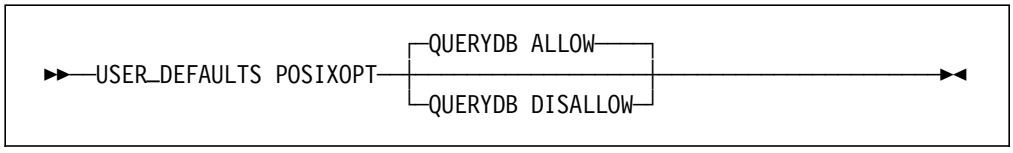
- A new OVM segment has been added to USER and GROUP profiles so that OpenEdition security information may be stored in the RACF database. VM will use the OpenEdition information defined in the RACF database instead of the information defined in the CP directory. In addition, POSIX program calls that request user and group information will return the RACF information as well.
- RACF also performs access checking for the Byte File System (BFS) on VM. Standard OpenEdition access checks are based on access bits associated with a file in the BFS. In addition, RACF audits file and directory access checks, and other OpenEdition events, based on LOGOPTIONS and AUDIT settings of several new classes.
- The RPIDIRCT EXEC has been enhanced so that it is now possible to process only OpenEdition-specific information, easing the migration of CP directory information into RACF.

  For more information on RACF, see *RACF General Information*, GC28-0722.

7.4.2 POSIXOPT QUERYDB

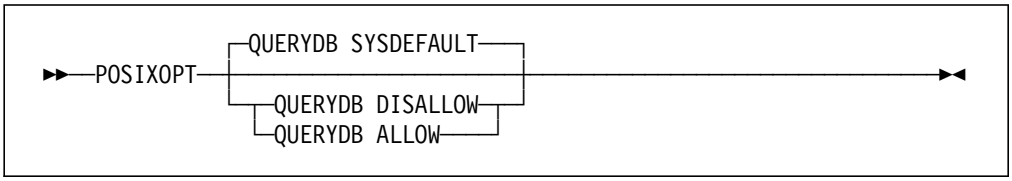
A POSIXOPT QUERYDB option on the USER_DEFAULTS system configuration file statement allows an installation to define whether all users on the system are permitted to query other users' user and group database information.

POSIXOPT QUERYDB System Configuration Statement



The system default can be overridden for users on an individual basis using the POSIXOPT directory control statement.

POSIXOPT QUERYDB Directory Statement



This authorization applies to the following POSIX functions. Their CSL routines (BPX) are listed as well.

`getgrnam` (BPX1GGN) Accesses the group database by name.

`getgrgid` (BPX1GGI) Accesses the group database by ID.

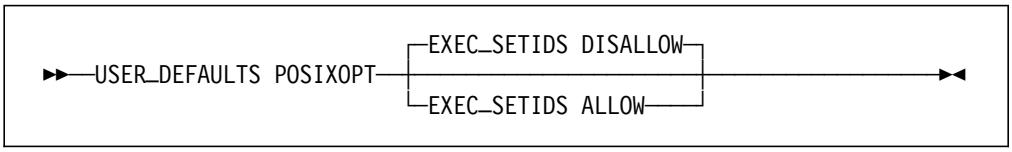
`getpwnam` (BPX1GPN) Accesses the user database by user.

`getpwuid` (BPX1GPU) Accesses the user database by user.

7.4.3 Exec Family of Functions

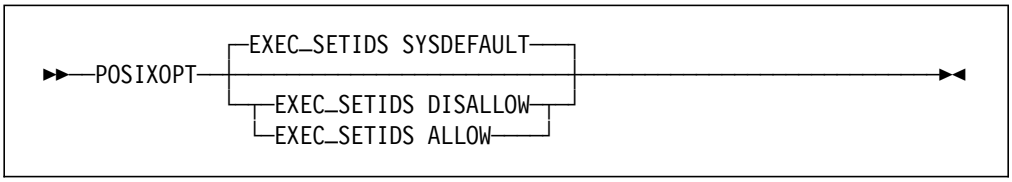
For any one of the *exec* family of functions, if the executable file has the set-user-ID mode bit set and the effective UID of the process is not the same as the file owner, or if the executable file has the set-group-id mode bit set and the effective GID is not the same as the file group, the caller must have CP authorization to execute these files. This authorization is defined on a system-wide basis in the system configuration file or on an individual user basis in the CP directory. It specifies whether a user with the appropriate file access permissions is allowed to execute a file that will result in a change of POSIX IDs.

POSIXOPT EXEC_SETIDS System Configuration Statement



The system default can be overridden for users on an individual basis using the POSIXOPT directory control statement.

POSIXOPT EXEC_SETIDS Directory Statement



In addition, the file server on which the object file resides must have this CP authorization to specify that the POSIX IDs for the caller should be changed. This permission is also defined in the CP directory. If either of these security checks fails, the existing process will be terminated, and the new file will not be invoked.

7.4.4 mailx Considerations

The `mailx` utility invokes another program, `/usr/lib/tsmail`, to transmit mail to other users, `tsmail` is a set-user-ID type of program. If your VM user ID is not authorized to run set-user-ID programs, then you cannot use `mailx` to send mail to other users, but other `mailx` functions will still work. As previously discussed, authorizations for set-user-ID programs are controlled in the CP directory and in CP's configuration file, SYSTEM CONFIG.

Appendix A. Shell Script Programming

A shell, acting as a command interpreter, allows the user to interact with the computer resources, such as programs, files, and devices. The user types the commands and the shell carries them out using system resources. It is through the shell that the OpenEdition user is expected to interact with the POSIX environment.

Most shells support a native programming language. Users can combine command sequences to create new programs. These programs are known as *shell scripts*. The shell scripts automate use of the shell as a command interpreter. Those familiar with the REXX or EXEC languages may some similarities.

If you frequently use the same six shell commands, you could include these six commands in a file, execute it, and avoid the need to retype the six commands every time you need information. Referencing this file, the shell will interpret the lines in the file, and execute each of them, as they were typed in the keyboard terminal. This is the most basic implementation of a shell script. Higher level scripts interact with the user, parse results, combine commands in pipes, read files, call subroutines, perform decision logic, and even support recursion.

The OpenEdition Shell and Utilities Feature for VM/ESA is, above all, a programmer′s interface. It has many general tools that can help any user or C programmer. Putting commands in a shell script has several advantages over typing the commands individually. Using the shell script you can:

- Reduce the number of errors.

  The more commands you type in the shell, the more opportunities you have to make a mistake. By putting them in a shell script, you reduce the risk of errors.

- Make it easy for other people to do your job.

  When a series of commands are in a shell script it becomes a repeatable procedure that is exploitable by even the most computer-timid.

- Reduce the amount of typing you have to do.

  By creating a library of commonly assembled commands into various shell scripts, all you need to do is enter the name of the script, instead of constantly retyping the commands.

For all of these reasons, you will find that using shell script makes your work enjoyable and more productive.

A.1 Basic Constructs

The shell has some advanced programming capabilities. Although shell script syntax is not as consistent as most of the conventional programming languages, its power and flexibility are comparable. Some aspects of the shell script programming are really extensions of the existent command environment, while others resemble the features of traditional programming languages. The use of variables, arithmetic expressions, and return values are some of the functions that could be used in a shell script. Similarly, attributes can be associated to variables, as well as exported to another shell script.

Shell scripts are normal files that live in the BFS. After you create a shell script, you need to be sure that the file containing the shell script has read and execute permissions. Use the chmod and unmask commands to set the permissions.

The following list defines the major functions of shell programming. In the sections that follow, the basic concepts that allow a shell script to function are discussed.

- I/O Mechanism

  More than one file can be opened at a time. Data can be read and written selectively.

- Menu Selection

  A shell script provides the structure to write programs that query the user through menus.

- Built-in arithmetic

  Shell script can perform arithmetic operations using the C language expression syntax.

- Substring operators

  You can generate substrings from values of shell variables.

- Array variables and attributes

  Strings can be converted to uppercase or lowercase. One dimensional arrays of strings or numbers can be used.

- Co-process facility

  The shell provides the capability to run one or more programs in the background, and to send and receive queries from the programs.

A.1.1 Variables

Like an alias, a variable is a name that has a value associated with it. When you want to use the value, you can use the name instead. A variable name consisting of one or more identifiers separated by a .(dot) or a _ character is called a varname. Assign values to variables using variable assignment commands. Variable assignment commands are of the form name=value. You cannot put spaces before or after the = sign. Also, the first character cannot be a digit, but you can use characters of any case.

A shell script runs like a separate shell session. By default, it does not share any variables with your current shell session. If you define a variable var in the current session, it is local to the current session, and any shell script that you call will not know var. To pass a variable and make it global, you can use the export command. The export command states that you want the variable var passed on to all the commands and shell scripts that you process in this session. export makes var global and known to all the commands and shell scripts in the session.

```
export MYVAR="street_number"
```

In this example, MYVAR will be known to all the commands and shell scripts, and it may be included in their tasks.

Value of Variables

Throughout the examples in this section, you will see the dollar sign followed by a variable name or expression, as in the following example:

```
${varname}
```

This expression obtains the value of the variable *varname*. You will often see this expression without the matching braces. This is not technically correct. For example, to read the tenth argument, $10 produces an error. ${10} is the correct way to extract the tenth argument of a shell script. The braces also allow you to use special characters as part of the variable name that may otherwise be interpreted by the shell.

Command Substitution

Direct assignment is just one way to get data into a variable. Using command line substitution, standard output from any command can be used as data for a variable. Wrap the command in parentheses and prefix it with a $ sign. For example, your shell script may want to use the output of the `ls -l` command for later use.

```
fileinfo=$(ls -l myshell.scp)
echo $fileinfo
```

String Lengths

There are many times where you want to check the length of a string. All you need to do is to prefix the variable name with a # sign, wrap it in braces and prefix it with a $ sign. The result is a number you can use for calculations or comparisons. For example, you may need to determine the length of a phrase, to make sure it will fit on your print label.

```
record="Sally Sells Sea Shells at the Park"
if [ 10 -lt ${#record} ]
then print "Sally has a large job title!"
else print "Sally is OK."
fi
```

We will cover the `if then` statement in section A.1.3, “Control Structures” on page 174.

Positional Parameters

You can define values for variables. The shell reads the shell script and tries to run the commands within it. But when the shell finds the $1 construct in the command, it goes back to the command line and obtains the first string following the name of the shell script on the command line. The shell replaces $1 with the string it reads in the command line. This type of construct is called a positional parameter.

Positional parameters have names 1, 2, 3, and so forth, meaning that their values are denoted by $1, $2,...,$N. There is also a positional parameter 0, whose value is the name of the script. Two special variables contain all of the positional parameters (except positional parameter 0): * and @. The difference between them is difficult to detect, but it is important, and it appears only when they are within double quotes.

$* is a single string that consists of all of the positional parameters, separated by the first character in the environment variable IFS (Internal File Separator), which is a space, tab, and new-line by default. For an example $@ is equal to $1, $2,...,$N, where N is the number of positional parameters in the line

command. The variable # holds the number of the positional parameters as a character string. All these variables are read-only, meaning that you cannot assign new values to them within scripts.

You can enclose, in double or single quotation marks, a $N construct in a shell script. When double quotation marks are used, the parameter is replaced by the appropriate value from the command line. When you use a single quotation mark to enclose a $N construct in the shell script, the $N is not replaced by the parameter value.

If you would like to show the name of the script you could code the following shell script:

```
echo $(basename $0)
```

Special Parameters

These are positional parameters starting with $ character. They may be used on the command line and in shell scripts.

They perform the following:

$@ Expands the complete list of positional parameters, starting with $1, and separates them with a single space.

$* Expands the complete list of positional parameters, starting with $1, and separates them with the first character of the value of an IFS shell variable.

$# The number of the positional parameters passed to this shell script. This number can be changed by several shell commands. This expression would return a 0 if there were no parameters passed to the shell script containing it:

```
echo $#
```

$? The exit status value returned by the most recently run command. The command echo $? returns the status for the most recently run operation or command.

$- The set of options that have been specified for this shell session. This includes options that were specified on the command line that started the shell, plus other options that have been set with the set command.

String Operators

When you reference a parameter, you can modify the value of its expansion by following the parameter inside the braces with one of the following string operators:

- A value to be used if the parameter is not set. If the default is to be used when the parameter is null, use a colon before the dash. In this example, if no parameters are passed to the shell script, a default parameter of compile is assigned to arg1, however, the value of $1 remains unchanged.

```
arg1=${1:-compile}
echo $arg1
echo $1
```

= This operator is similar to the previous one. The difference is that if the parameter does not currently have a value, then a value is assigned to the parameter, and the new value for the parameter is used. Here the parameter must be a variable, not a positional parameter. In the example,

$1 is assigned the value of compile when no parameters are passed to the shell script:

```
arg1=${1:=compile}
echo $arg1
echo $1
```

? A message to be displayed as a standard error if the parameter is not used. Use a colon before the question mark to cause the message to be displayed if the parameter is null or not set:

```
echo ${1:?"You must supply a parameter"}
```

The smallest leading portion of the value matching the pattern following # is discarded. If you execute this shell script in your home directory, the /home/ portion of the path will be removed:

```
cdir=$(pwd)
echo ${cdir:#/home/}
```

The largest leading portion of the value matching the pattern following ## is discarded. This example will strip leading zeros:

```
rate="0000000001.118"
echo ${rate:##*0}
```

% The smallest trailing portion of the value matching the pattern following % is discarded. This operation works similar to #, but from the other direction:

```
${parameter:%pattern}
```

%% The largest trailing portion of the value matching the pattern following % is discarded. This operation works similar to ##, but from the other direction:

```
${parameter:%%pattern}
```

Regular Expressions

Regular expressions are a form of shorthand pattern matching that has become a vital part of computer languages. They are similar to wildcard matches, but much more powerful. The syntax follows the form used in tools such as awk and grep. Below is a sample that will compare a string with a range of values:

```
if expr "b" : '[abc]'
then print "match"
else print "no match"
fi
```

The brackets must appear in quotes to keep them from being expanded. The condition located after an if-then clause is called a *test*. In fact, the word test may be used in place of the square brackets. A test condition after an if-then using a regular expression will not work. expr handles regular expressions correctly.

A.1.2 Shell Script Return Values

Each command has a return value. The return value of a command that terminates for a reason other than the receipt of a signal is a number from 0 to 255. The following is a brief description of the return values:

0 Normal exit. When it is used in a conditional or iteration command, this value represents True. Any other value means that the command returns False.

1-125 Failure.

126	A command was found but cannot be executed.
127	A command could not be found.
128-255	Additional failures.
256 and above	A command has terminated or exited due to a reception of a signal. You can use the `-l` option of the kill command to determine the name of the signal that caused the command to exit, or subtract 256 to get the number.

Arrays

So far we have seen two different types of variables: character strings, and integers. The third type is known as an *array*. An *array* is like a list of things. An array is a variable that can store multiple values, where each value is associated with a subscript or index. There are two types of arrays: indexed arrays and associated arrays. The subscript for an indexed array is an arithmetic expression and the subscript for an associated array is an arbitrary string of characters. REXX supports only arbitrary strings.

You can specify an array using the standard shell variable assignment syntax, putting the array index between brackets [ ]. For example:

```
nickname[2] = gus
nickname[3] = rod
nickname[4] = mik
```

In this example the values `gus`, `rod`, and `mik` are put in each of the elements of a variables nickname, with indexes 2, 3, and 4, respectively. With regular shell variables, values assigned to array elements are treated as character strings unless the assignment is preceded by the `let` statement.

The second way to assign values to an array is with the `set` statement, for example:

```
set -A nickname gus rod mik ...
```

As you can see, `set` creates the array `nickname` and assigns the value of `gus` to the `nickname[1]` index, the value of `rod` to `nickname[2]`, and so forth. This is more convenient for loading up an array with an initial set of values. You can extract the value from an array using the syntax *${arrayname[i]}*. For the previous example, `${nickname[3]}` would have the value of `mik`.

In a shell script, you can define only a one-dimension array. They are limited to 1024 elements, and the index may start at 0. Arrays of arrays are not allowed.

Typeset

A feature of the shell relating to the kind of values that variables can hold is the typeset command. `typeset` is used to specify the data type of a variable.

The options available fall into two basic categories:

- String formatting operations, such as right and left justification, truncation, and letter case control
- Type and attribute functions

Another characteristic of `typeset` is that if it is inside a function definition, without any argument, all the variables involved become local to that function, in addition to any other properties that they may have.

Other attributes may specify how the variable's value is displayed when the variable is expanded. The following list summarizes these attributes:

-Ln Left justify. Remove leading blanks, and if *n* is given, fill with blanks or truncate on right to length *n*.

-Rn Right justify. Remove trailing blanks, and if *n* is given, fill with blanks or truncate on left to length *n*.

-Zn Justify, except add leading 0's instead of blanks if needed. Specify either -RZn for right justification or -LZn for left justification.

-l Converts letters to lowercase.

-u Converts letters to uppercase.

You can also specify the read-only attribute -r. It allows you to mark a variable that is being exported. This means that the value of the variable cannot be changed by subsequent commands.

To display the currently defined variables and their attributes, use the -x option. This will display all the variables currently defined for export.

The -f option has a set of suboptions. They are:

-f With no arguments, prints all function definitions.

-f fname Prints the definition of function *fname*.

+f Prints all function names.

-ft Turns on trace mode for a named function.

+ft Turns off trace mode for a named function.

-fu Defines given name as an auto-loaded function.

Finally, if you type typeset without any argument, you can see a list of all currently defined variables and their settings. For example, Figure 100 on page 174 is the output of the execution of typeset without any commands.

```
 maintmo /home/maintmo/ $

typeset
@
-i ERRNO
-x HOME
IFS
-i LINENO
-x LOGNAME
-x MAIL
-i MAILCHECK
-x MAILMSG
-x OLDP
-i OPTIND
-x PATH
-r PPID
PS1
PS2
PS3
PS4
PWD
-i RANDOM
-i SECONDS
-x SHELL
-x TZ
-x _
-x _EDC_KEEP_EMSG

 maintmo /home/maintmo/ $
```

Figure 100. Typeset Example

A.1.3 Control Structures

The shell provides facilities similar to those found in programming languages. These facilities include the *control structures*, which are related to programming flow control. They are:

- Test Commands

  They are commands that test if something is true. They examine the nature of a file, if a path name exits, if it is readable, and so on.

- Do

  As in other programming languages, the shell interprets a series of commands in an ordered way. The do expression executes a series of commands in a script as if they were a single routine. This allows you to have multiple routines inside a single script or file. The end of this condition is marked by the expression done.

- If Conditional

  Runs a sequence of commands if a particular condition is met. It has the form of:

  ```
  if condition
  then command1
  elif condition
  then command2
  fi
  ```

  The end of the commands is indicated by fi.

The condition can be met with a test statement, but a better way is to use brackets ([ ]) to enclose the conditional test, as in this example:

```
record="billy"
if [ "bill" = $record ]
then print "match"
else print "no match"
fi
```

- Loops

  The shell allows interactive execution of a series of commands based on a condition. Depending on the way you request the test, the shell could repeat a series of commands several times until another condition is reached.

  The `while` **Loop** The `while` loop repeats one or more commands while a particular condition is true. The format of the while loop is:

  ```
  while condition
  do commands
  done
  ```

  The shell first tests to see if condition is true, and if it is, runs the commands. Then, the shell tests the condition again, and if it is still true, runs the commands again and so on, until the condition is found to be false.

  The `for` **Loop**

  You can use the `for` to repeat a sequence of commands once for each item in a specified list. Before a loop begins, the shell expands the list of words that you specified following the reserved word `in`.. These words are expanded just as the shell expands command words. The body of the `for` command is executed once for each argument in the expanded list.

  ```
  for name in list
  do commands
  done
  ```

  The parameter *name* should be a variable. The parameter list is a *list* of strings separated by spaces. The shell assigns the loop index variable `i` in the following example:

  ```
  for i in * # For each file in the working directory
  do if  [[ -d $i ]]
  then print -r -- "$1"   # Print subdirectory names
  fi
done
  ```

- Combination of Structures

  Functions of the different structures may be combined in a single program. You can also combine control structures by nesting them, that is, putting one inside the other. For example:

```
for file in $ (find . -name "*.c")
do
  if test $file -ot $1
  then
  echo $file
  c89 -c file
  fi
done
```

- Case statement

  Another control structure is case. case finds which specified pattern matches a given word, and executes the command list associated with that pattern. case can also be used to test simple values like integers and characters.

  The syntax of the case statement is as follows:

```
case expression in
   pattern1)
      statements ;;
   pattern2 )
      statements ;;
   ...
```

  Any of the patterns can actually be several patterns separated by pipe characters (|). If *expression* matches one of the patterns, its corresponding statements are executed. If there are several patterns separated by pipe characters, the expression can match any of them in order for the associated statements to run. The patterns are checked in order until a match is found; if none is found, then nothing happens.

A.1.4 Open and Closing Files

Use exec without arguments to open and close files in the current environment. Use the I/O redirection syntax to specify the files that you want to open or close. You can open or close file descriptors from 0 to 9. The shell sets the close-on-exec bit on file descriptors from 3 to 9.

Use the I/O redirection without a command name to create a file or remove the contents of an existing file. The shell opens or creates the file and then closes it.

The parameter $ expands to the process ID. Each script has its own unique ID. Use $$, the value of parameter $, within a path name to generate the name of a temporary file to ensure that the temporary file name is unique. Let's look at the following example:

```
exec 3 < mik  # Opens mik for reading, as file descriptor 3
exec 3<&-     # Closes file descriptor 3
>/tmp/mik$$   # Creates a temporary file
exec 3<> mik  # Opens mik for reading and writing, as file descriptor 3
```

The shell processes certain file names specially when specified after a redirection operator. A file name of the form */dev/fd/number* is treated as an indication to use the file descriptor number.

A.1.5 Reading from Terminals and Files

By default, the data in the shell is line-oriented. The shell does not process its input until it reads the new-line character. You can use read to read from any open file. read will read one line at a time. Specify read -r when reading from files. The \ character is the continuation character.

As a command begins running, it has access to three files:

- It can read from its *standard input* file. By default the standard input is the keyboard.
- It writes to its *standard output*. By default the standard output file is the console.

- It writes error messages to its *standard error* file. By default the standard error file is the console.

In OpenEdition Shell and Utilities Feature for VM/ESA, the names for these files are:

- `stdin` for the standard input file
- `stdout` for the standard output file
- `stderr` for the standard error file

The shell references these files by their file descriptors or identifiers:

- 0 for `stdin`
- 1 for `stdout`
- 2 for `stderr`

A.1.6 Writing to Terminals

Use the `print` and `echo` commands to display lines on a terminal and to write lines to a file. `echo` writes to the standard output or console, as specified by the arguments. `echo` accepts the C-style escape sequences, for example, `\n` for a new line.

In OpenEdition Shell and Utilities Feature for VM/ESA, you have two options with the `print` command. You can print to the standard output or file, using the `print` command. But if you need to format this output, you can use the `printf` command; `printf` allows you to use a series of flags to format the output. The `print`, `printf`, and `read` commands are well-described in *OpenEdition for VM/ESA: Command Reference*, SC24-5728.

A.1.7 Shell Functions

A *shell function* is similar to a subroutine in other programming languages: it is a sequence of commands that do a single job and that are callable. Usually a function is used to do an operation that you do frequently in a shell script. You have to define it in the shell script to call it.

You can define a function in either of these two ways:

```
function functname {
   shell commands
}
or
functname ()  {
   shell commands
}
```

There is no difference between the two.

To call a function inside a shell script, use the `td` command. The function helps organize large scripts into smaller blocks of subroutines, aiding the development of complex tasks. The function differs from a shell script in that it can run in the current environment and, therefore, it can share variables and other effects with the script that invokes it.

You can use a `return` command to return to the calling statement. If you do not use `return`, the function returns after the shell processes the last command. If

you call from an external script and you do not use a return, you will never return to the original script and follow the correct nested order of execution. You also can use the exit command to terminate the function. Do not use exit unless you want to terminate the current script.

The following example could be used in the /etc/profile to create a global function that allows users to call XEDIT from within the shell using a minimum of typing:

```
function xedit {
  cms "XEDIT '$1' (NAMETYPE BFS"
  return
  }
```

A.1.8 Combining Commands

There are several ways to combine many commands on a single line. You can run a series of commands one after the other:

- Using a semicolon (;)

  The shell lets you enter several commands on the same command line with the use of a semicolon character to separate the commands.

  ```
  cd mydir ; ls -l
    or
  cd mydir ; test.scp
  ```

- Using &&

  When stringing together more than two commands, you may want to control the running of the second command based in the results of the first command. You can use the && between two commands to run the second only if the first completes successfully. You have to leave a space on either side of the && operand.

- Using ||

  When you want to execute a second command no matter how the first command completes, use the || operand.

Or you can run more than one command concurrently, using a pipe or a filter with a pipe.

- Pipe (|)

  The output of one command can be *piped in* as input to the next command. Two or more commands linked by a pipe are called a *pipeline*. To use a pipe, enter the commands in the same line and separate them by the pipe character **|**.

  ```
  ls | wc -l
  ```

  In this example, you list the contents of a directory, and this output will be the input for the word count command, which will return the number of files that exist.

A.1.9 Debugging a Shell Script

The shell has a few features that help in the debugging of shell scripts. You can check the syntax of a shell script without actually executing it. The OpenEdition Shell and Utilities Feature for VM/ESA allows you to find where the simple errors are located; however, you may need to see the commands as they are run. For this, use the set -o xtrace command. An example is shown in Figure 101.

```
00000 * * * Top of File * * *
00001 function testf  {
00002    set -o xtrace
00003    if test $# -eq 0
00004    then echo "testf: Enter an argument"
00005    elif [ -f $1 ]
00006    then echo "testf: "$1"  is file"
00007    elif [ -d $1 ]
00008    then echo "testf: "$1"  is a directory"
00009    else echo "testf: "$1"  not exist"
00010    fi
00011 }
00012 testf $1
00013 * * * End of File * * *


 maintmo /home/maintmo/ $
testf test          1
+ test 1 -eq 0
+ [ -f test ]
+ [ -d test ]
+ echo testf: test  not exist
testf: test  not exist

 maintmo /home/maintmo/ $

testf test          2
 testf: test  not exist

  maintmo /home/maintmo/ $
```

Figure 101. Example of a Debugging Aid

At the top of Figure 101, the -o option of xtrace is set and the output is shown as lines are run (1). The last output shows the result of the shell script without this option (2).

You can also use two other settings. The verbose setting allows you to see the shell script input lines as they are read. The noexec option does not run the commands; it only checks for syntax errors. You can set these options at the command ny typing them before you execute the shell.

A more sophisticated set of debugging aids are called *signals*. You can use the trap command to select code that runs when a particular signal is sent to your script. This permits the interception of certain types of exception conditions. The two signals that you can use in OpenEdition Shell and Utilities Feature for VM/ESA are exit and err.

The signal err allows you to run code whenever a command in the shell script or function exits with non-zero status. The trap code for err can take advantage of built-in variable ?, which holds the exit status of the previous command. It survives the trap and is accessible at the beginning of the trap code.

The exit trap will run code after the calling script exits. An exit trap occurs no matter how the script or functions exits, whether normally by using a regular exit or abnormally by receiving a signal.

A.1.10 Shell Extensions

In a VM/ESA environment you can use REXX with the shell. REstructure eXtended eXecutor is a high-level interpreted language that is native to the CMS environment, and is similar to shell scripts in many ways.

In the shell, the use of the cms command allows you to execute any CP or CMS command including REXX EXECs.

A.2 Sample Shell Scripts

This section contains shell script examples.

A.2.1 FS

The FS shell script will search for multiple files entered as an argument.

```
#
# File Search
# Looks the address of a given file or multiple files
#

if [ $# -eq 0 ]              # Check for an argument
then                         # if not

 cms 'VMFCLEAR'              # Let's clear the screen
 echo                        # Display the usage of the script
 echo "The use of the command is:  fs fname fname ... fn"
 echo

else                                 # if there is an argument;
 names="( -name $1"                  # Make the variable names
 shift                               # Change if we got more than one
 for flist                           # loop for multiple arguments
  do                                 #
  names="$names -o -name $flist"     # look for multiple arguments
                                     # using the "OR" option
  done                               # Finish
 names="$names )"                    # close the variable

 cms 'VMFCLEAR'                      # Clear the screen
 echo
 echo "File(s) exist(s) in:"         # Display the msg
 find / $names -print 2> /dev/null | pr -t  # Execute the find
                                            # send it through the pipe
                                            # and print without headers
fi                                   # End
```

Figure 102. Multiple File Search

A.2.2 TS

Tree Search is a tool that mimics the function of the `ls -lRd` command. It will display all the files in all the sub-directories on a given directory.

```
#
# TS  Tree Search
#

if test $# -gt 1
then
 echo "usage: ts [dir]"
 exit 1
fi
if test $# -eq 1
then
 pwdir=$1
else
 pwdir=pwd
 fi

echo
for dir in find $pwdir -type d -print
do
 echo
 echo "$dir"
 ls -lsi $dir
 echo
done
```

Figure 103. Tree Search

A.2.3 BFS

Call the BFSLIST REXX EXEC using a shell script to choose the directory.

```
#
# Shell Script to call BFSLIST EXEC
#
if [ $# -eq 1 ]        # Check for arg
then
   pwdir=$1                      # Use path name supplied
else
   pwdir=pwd                   # Use current directory
fi
cms bfslist $pwdir               # Execute BFSLIST EXEC
```

Figure 104. BFS Shell Script

A.2.4 TESTFILE

This script determines the type of a file called as an argument. It uses the test statement.

```
function testf  {                                          # Create a function
   if test $# -eq 0                                        # Test for an argument
   then echo "testf: Enter the name for file name test"    # if not request one
   elif [ -f $1 ]                                          # if is, tell it, if not next
   then if [ -x $1 ]                                       # Test for a File
         then echo "testf: "$1"  is an executable file"
         else echo "testf: "$1"  is a regular file"
        fi
   elif [ -L $1 ]                                          # Test for a Symbolic Link
   then echo "testf: "$1"  is a SYMBOLIC LINK"
   elif [ -d $1 ]                                          # Test for a Directory
   then echo "testf: "$1"  is a directory"                 # if is, tell it, if not next
    # end
   else echo "testf: "$1"  not exist"
   fi
}
testf $1                                                   # call the function and pass the
                                                           # argument
```

Figure 105. TESTFILE - (Multiple File Search)

A.2.5 COUNT

Count will tabulate how many files are in a directory.

```
#
# Count the number of files of a given directory
#
curdir=$(pwd)    # gets the current directory for return
cd $1            # change to the given directory
ls | wc -l       # count the files by the pipe
cd $curdir       # back again to the root
#
```

Figure 106. Count Shell Script

Appendix B. Program Source Code

This section contains the source code for the large programs discussed throughout this publication.

B.1 CSLPSCAN CP Directory POSIX Option Reporting Tool

The REXX program shown in Figure 107 on page 184 is an example of how to use some OpenEdition CSL calls. This program produces a report of the users assigned POSIX attributes in the CP directory.

Because some readers of this publication might not be familiar with REXX and VM, this program will be explained in detail.

```
/*--------------------------------------------------------------------*/
/* C S L P S C A N                                                    */
/*                                                                    */
/* Form:     SG24-4747                                                */
/*                                                                    */
/* Format:   cslpscan <gname | gid>                                   */
/*                                                                    */
/* Process:  The CSL service BPX1GGI or BPX1GGN is used in order      */
/*           to list the members of a particular group.               */
/*           Also for each member POSIXINFO data is displayed.        */
/*                                                                    */
/* Notes:    - If no arguments passed, default is your user gname.    */
/*--------------------------------------------------------------------*/
Trace 'O'                           /* trace setting                */ 1
Address 'OPENVM'                    /* address setting              */ 2
Signal on ERROR                     /* in case of errors            */ 3
@.=''                               /* global variables init        */ 4
$.=0                                /* global switches init         */ 5
Parse Arg @.@arg .                  /* parameters passed, lowercase!*/ 6
If (@.@arg='?') | (Translate(@.@arg)='HELP') /* help requested?     */ 7
           then Call Rout_help      /* display exec header comments */
/*-----------------------------------*/
If $.@ps0=0 then Call Rout_init     /* initialization & setup       */ 8
If $.@ps0=0 then Call Rout_main     /* main process                 */ 9
If $.@ps0=0 then Call Rout_term     /* termination & cleanup        */ 10
/*-----------------------------------*/
Exit $.@ps0                         /* e x i t  (only one)          */ 11
/*--------------------------------------------------------------------*/
Rout_init:                          /* Routine: Rout_init           */
Procedure expose @. $.              /* procedure                    */ 12
Parse Source . . @.@sfn .           /* source file name             */ 13
If @.@arg = '' then 'BPX1GID @.@arg'  /* get our own GID (as default) */ 14
Return                              /* r e t u r n                  */ 15
/*--------------------------------------------------------------------*/
Rout_main:                          /* Routine: Rout_main           */
Procedure expose @. $.              /* procedure                    */
group = @.@arg                      /* get the requested parameter  */ 16
groupLen = Length(group)            /* length of group (if GNAME)   */ 17
If Datatype(group,'NUM')            /* check if GID or GNAME        */ 18
 then 'BPX1GGI group return_value return_code reason_code'            19
 else 'BPX1GGN groupLen group return_value return_code reason_code'   20
If return_code=0                    /* function completed ok?       */ 21
 then Call Sub_BPXYGIDS return_value                                  22
 else Call ERROR_Openvm return_code reason_code 'BPX1GG_'             23
Return                              /* r e t u r n                  */
/*--------------------------------------------------------------------*/
Rout_term:                          /* Routine: Rout_term           */
Procedure expose @. $.              /* procedure                    */
Say @.@sfn':' Copies('_',70)        /* separator                    */ 24
Return                              /* r e t u r n                  */
/*--------------------------------------------------------------------*/
Rout_help:                          /* Routine: Rout_help           */
Procedure expose @. $.              /* procedure                    */
Address 'COMMAND' 'VMFCLEAR'        /* clear the screen             */ 25
Do i=1 while Left(Sourceline(i),2)='/*'/* 1 Do check if header comms.  */ 26
 Say Sourceline(i)                  /* dump prolog                  */ 27
 end                                /* 1 level, Do end.             */
$.@ps0=911                          /* set return code to exit      */ 28
Return                              /* r e t u r n                  */
```

Figure 107 (Part 1 of 2). Sample EXEC to Demonstrate BPX CSL Calls

```
/*--------------------------------------------------------------------*/
ERROR:                                 /* Routine: ERROR              */ 29
If Arg() > 0 then rc=Arg(1)            /* get RC if directly passed   */ 30
$.@ps0=rc                              /* error rc                    */ 31
Say @.@sfn': Error occurred in line' SIGL', RC:'rc                       32
Return                                 /* r e t u r n                 */
/*--------------------------------------------------------------------*/
ERROR_Openvm:                          /* Routine: ERROR_Openvm       */
Arg OV_return OV_reason Routine_name   /* arguments passed            */ 33
Call DMSOVB(OV_return OV_reason 'CSL 0' Routine_name 0)                  34
$.@ps0=OV_return                       /* error rc                    */
Say @.@sfn': Error occurred in line' SIGL', return_code:' OV_return
Return                                 /* r e t u r n                 */
/********************************************************************/
Sub_BPXYGIDS:                          /* Routine: Sub_BPXYGIDS       */
Procedure expose @. $.                 /* procedure                   */
Numeric digits 10                      /* 'default' id is a big number */ 35
/*-----------------------------------*/
Parse Arg data 1 G_len +4 data         /* data from getgrnam ! getgrgid*/ 36
G_len=C2D(G_len)                       /* length of group name        */ 37
Parse var data 1 G_name +(G_len) data  /* get the group name          */ 38
Parse var data 5 GID +4 count +4 data  /* get the group id            */ 39
GID=C2D(GID)                           /* conversion                  */ 40
count=C2D(count)                       /* conversion                  */
Say @.@sfn':' Copies('_',70)           /* separator                   */
Say @.@sfn': Information for group' Strip(G_name)'('GID') ',             41
         'Total number of members:' count'.'
Say @.@sfn':' Copies('_',70)           /* separator                   */
/*-----------------------------------*/
Say @.@sfn': Username        UID        GID IWDIR    IUPGM'
Say @.@sfn': -------- ---------- ---------- --------' Copies('-',30)
Do i=1 to count                        /* 1 Do handle all members     */ 42
 Parse var data 1 M_len +4 data        /* first/next member           */ 43
 M_len=C2D(M_len)                      /* length of member name       */
 Parse var data 1 M_name +(M_len) data /* get the member name         */
 'BPX1GPN M_len M_name return_value return_code reason_code'             44
 If return_code=0                      /* function completed ok?      */ 45
  then Do                              /* 2 Do parse BPXYGIDN         */
       Parse var return_value 1 U_len +4 return_value                    46
       U_len=C2D(U_len)                /* length of user name         */
       Parse var return_value 1 U_name +(U_len) return_value
       Parse var return_value 5 UID +4 return_value
       UID=C2D(UID)                    /* conversion                  */
       Parse var return_value 5 GID +4 return_value
       GID=C2D(GID)                    /* conversion                  */
       Parse var return_value 1 D_len +4 return_value
       D_len=C2D(D_len)                /* length of working directory */
       Parse var return_value 1 D_name +(D_len) return_value
       Parse var return_value 1 P_len +4 return_value
       P_len=C2D(P_len)                /* length of user program name */
       Parse var return_value 1 P_name +(P_len) return_value
       Say @.@sfn':' Left(U_name,8),  /* display member information   */ 47
                     Right(UID,10),
                     Right(GID,10),
                     Left(P_name,8) D_name
       end                             /* 2 level, Do end.            */
  else Call ERROR_Openvm return_code reason_code 'BPX1GPN'                 48
 end                                   /* 1 level, Do end.            */
Return                                 /* r e t u r n                 */
```

Figure 107 (Part 2 of 2). Sample EXEC to Demonstrate BPX CSL Calls

1 The *Trace* instruction controls tracing in a REXX program. In this case tracing is *Off*. You can use *Trace Results* to trace all clauses before execution. This setting is recommended for general debugging of your programs.

2 The *Address* instruction changes the destination of commands. The default is CMS, but because of the nature of this program you want all commands to address OPENVM functions.

3 SIGNAL ON ERROR is used here to control the trapping of error conditions. A condition is raised when a command indicates an error condition on return (for example a non-zero *RC*). When the error condition occurs, control passes to the label ERROR where you can handle the problem in any appropriate way.

SIGNAL can also be used to cause a change in the flow of control, just like the GOTO instruction.

4 In REXX you have variables, compound variables, and *stems*. In this case **@.** makes all possible variations a null value. For example, when you use @.var1, you do not have to initialize it because it is already null.

5 This is the same case as in the previous example. This program tries to differentiate between *general variables* and *switching variables*.

6 Parameters passed to this program are handled with the Parse instruction. In this case the instruction functions like the Arg instruction. You use Parse because you do not want the arguments being translated to uppercase (remember group names in POSIX are case sensitive). Notice the *period* after the instruction. This will remove any extra leading or trailing blanks from the passed parameters.

Now why are you naming this variable **@.@**arg and not just @.arg? Imagine you have a variable called @.color. If you also have a plain variable called color and it contains the value BLUE, next time you tried to use your variable @.color you would be surprised to see it had become @.BLUE.

By using **@.@**color you avoid this kind of problem because you never use @color as a variable (all your @ variables are always *stem* variables). Additionally, if you are a REXX Compiler user, the cross-reference listing provides a handy method to control all your global variables.

7 Now the program checks if the user wanted some help. It translates the argument to uppercase to make sure no matter how you enter your help request (HELP, Help, heLP), it will be honored.

8 This is the core of the program containing the *Input, Process* and *Output*. In this case the *initialization* phase is called.

9 The *main process* phase is invoked here. The **$.@**ps0 variables (used here as *program status* variables) ensure that the routines will only be executed while everything is running without error. Otherwise the program falls through until it eventually reaches EXIT.

10 *Termination* is here.

11 The program exits. As you remember from your *basic* structured programming course, a good program (or complex system, or small subroutine) should only have *one entry point* and *one exit*, with no exceptions.

12 All subroutines in this program use PROCEDURE to manage the scope of variables. You do not have to worry about using variables already used by the caller (imagine changing a variable that is being used as a loop control variable in the caller subroutine).

13 Parse Source gets the name of your executing EXEC when called this way. You then use this name in all the messages produced by your program. Even when a program begins to produce errors, at least you will be able to tell *who* is reporting errors. Another advantage is that the program can be renamed without changing a constant defined within.

14 This program will default to your own group, if none is specified. BPX1GID is the first OPENVM routine used in this program. This *getgid* gets the real group ID (GID) of the calling process. As you are the calling process, it gets your own GID.

15 RETURN returns control to the point of invocation.

16 You assign the passed parameter to the variable group.

17 The Length function returns the length of the group. This is a required parameter for the BPX1GGN function.

18 Now is the time to check if your program got called with a *group name* or a *group ID*. This Datatype function checks if the parameter is numeric.

19 In this case the groupLen variable was not required.

20 BPX1GGN is the function that will access the group database by name. Remember that you really need the POSIXOPT QUERYDB in your CP Directory entry to execute these functions.

21 Was the function successful? return_code=0 means everything was fine. The flow of control was not altered as dictated by SIGNAL ON ERROR, because this instruction only reacts to the contents of *RC*. It has nothing to do with variable return_code.

22 You can now further process the data returned by the OPENVM function. You invoke subroutine Sub_BPXYGIDS with the returned value as a parameter.

23 There is some error with the executed OPENVM function. In this case you call your error handler subroutine. Besides passing the return_code and reason_code as parameters, you give the name of the offending function. In this case it is either BPX1GGI or BPX1GGN.

24 This is just a separator. This particular invocation of the Copies function returns 70 underscore characters that will be displayed at your terminal by the Say instruction.

25 VMFCLEAR is used here to clear the screen. This is a convenient way to begin your HELP text at the top of the screen.

26 Sourceline returns the *n*th line in your program here. Basically, the program loops, displaying your program header lines (in this case, acting as HELP text) *while* the first two characters are */** (the REXX begin comment indicator).

27 All HELP text qualifying lines are now displayed.

28 You only want to display your HELP text, so you set $.@ps0 to a non-zero value. As a result **8**, **9**, and **10** will never get executed.

29 This is the ERROR routine that **3** was talking about. This routine has no Procedure instruction because you normally get here as an altered flow of control and Procedure expects to be the first instruction executed in a routine (this normally happens when you *call* a subroutine).

30 You can also call this routine with CALL. In that case there is no *RC* available (SIGNAL was never triggered) and you have to provide your own *RC*. This instruction is testing for the presence of an argument. If it finds one, it assigns it to *RC*.

31 Here $.@ps0 is being set making the program stop because all the following routines test this variable before they are executed.

32 Following any transfer of control due to a CALL or a SIGNAL, the program line number of the clause causing the transfer of control is stored in the special variable *SIGL*. This indicates where the problem is coming from.

33 These are the parameters you can get from either **23** or **48**.

34 The DMSOVB EXEC (an unsupported interface) will analyze the passed parameters (*return_code* and *reason_code*) and display an appropriate error message. Notice that not all combinations are covered in DMSOVB, and it will display message number 2134 when no match is found.

35 As you already know, GIDs and UIDs can be as large as four billion. This is 10 digits, but REXX defaults to 9. Hence this instruction is required to process such extremes.

36 BPXYGIDS is actually a DSECT. If you are not familiar with assembler, it is like the definition of a record's fields. In this case the four first bytes contain the length of the group name.

37 This interface is not REXX-oriented. The returned value must be converted to decimal to process it in REXX.

38 Now you can use this length to obtain the group name by cutting field by field, leaving *data* ready for the next parsing.

39 GID is obtained. The DSECT mentions this as always being length four, so you do not need to parse the length.

40 This GID needs also to be converted to decimal.

41 Display the total number of members for this group.

42 Here you are going to process all of those members, one by one.

43 Perform the same steps as before to get the member name length.

44 You want to get *more* information about each member and you use BPX1GPN (Access the user database by user name). Basically its invocation is similar to BPX1GGN, but it uses a different DSECT (BPXYGIDN) to parse the returned values.

45 If everything is correct, then you can parse *return_value*.

46 Perform the same operations as before; get the length and then convert it to decimal.

47 Finally you display formatted information for each member of the group.

48 If BPX1GPN gives an error, follow the same actions as in **23** .

B.2 DIRPSCAN, a Directory Reporting Tool

This sample program can be used to create a report of the POSIX options defined in your CP directory.

```
/*--------------------------------------------------------------------*/
/* D I R P S C A N                                                    */
/*                                                                    */
/* Form:     SG24-4747                                                */
/*                                                                    */
/* Format:   dirpscan <fn ft fm>                                      */
/*                                                                    */
/* Process:  Information is extracted from the USER directory source  */
/*           file (supposed to be in monolithic format)               */
/*           Data from POSIXGROUP, POSIXINFO, POSIXGLIST and POSIXOPT */
/*           statements is extracted and listed accordingly.          */
/*           GIDs are translated to group names and a warning message */
/*           is issued if duplicated UIDs are found.                  */
/*                                                                    */
/* Notes:    - If no arguments passed, default is USER BACKUP *       */
/*--------------------------------------------------------------------*/
Trace 'O'                             /* trace setting                */
Address 'COMMAND'                     /* address setting              */
Signal on ERROR                       /* in case of errors            */
@.=''                                 /* global variables init        */
$.=0                                  /* global switches init         */
Parse Upper Arg @.@arg                /* parameters passed            */
If (@.@arg = '?') | (@.@arg = 'HELP')  /* help requested?             */
           then Call Rout_help        /* display exec header comments */
/*-----------------------------------*/
If $.@ps0=0 then Call Rout_init       /* initialization & setup       */
If $.@ps0=0 then Call Rout_main       /* main process                 */
If $.@ps0=0 then Call Rout_term       /* termination & cleanup        */
/*-----------------------------------*/
Exit $.@ps0                           /* e x i t  (only one)          */
/*--------------------------------------------------------------------*/
Rout_init:                            /* Routine: Rout_init           */
Procedure expose @. $.                /* procedure                    */
Parse Source . . @.@sfn .             /* source file name             */
fileid = Space(@.@arg Subword('USER BACKUP *',1+Words(@.@arg),3),1)
'PIPE <' fileid '| STEM @.@direct.'    /* read the source directory   */
Return                                /* r e t u r n                  */
/*--------------------------------------------------------------------*/
Rout_main:                            /* Routine: Rout_main           */
Procedure expose @. $.                /* procedure                    */
```

```
If $.@ps0=0 then Call Sub_groups       /* process groups               */
If $.@ps0=0 then Call Sub_users        /* process users                */
Return                                 /* r e t u r n                  */
/*--------------------------------------------------------------------*/
Rout_term:                             /* Routine: Rout_term           */
Procedure expose @. $.                 /* procedure                    */
@.@DISPLAY.0=$.@Display                /* number of lines              */
'PIPE STEM @.@Display.',               /* get the in storage data      */
   '| SORT 20-29',                     /* sort it by UID               */
   '| CONSOLE',                        /* glorious output coming up    */
   '| UNIQUE  20-29 MULTIPLE',         /* get duplicated UIDs          */
   '| NLOCATE 29-29 / /',              /* avoid POSIXOPT only users    */
   '| STEM dup.'                      /* save them here               */
If dup.0 = 0                           /* all right, no duplicates     */
 then Nop                              /* all done, bye (for now)      */
 else Do                               /* 1 Do inform about dups       */
       Say @.@sfn':' Copies('-',70)    /* separator                    */
       Say @.@sfn": The following userid's are found to have duplicated",
                   "GID's"
       Say @.@sfn": It is strongly recommended to have only unique GID's"
       Say @.@sfn':' Copies('-',70)
       'PIPE STEM dup.',               /* get the in storage data      */
          '| SPECS 1-29 1',            /* display only userid & uid    */
          '| CONSOLE'                  /* display users with dup. gid  */
       end;                            /* 1 level, Do end.             */
Return                                 /* r e t u r n                  */
/*--------------------------------------------------------------------*/
Rout_help:                             /* Routine: Rout_help           */
Procedure expose @. $.                 /* procedure                    */
'VMFCLEAR'                             /* clear the screen             */
Do i=1 while Left(Sourceline(i),2)='/*'/* 1 Do check if header comms.  */
 Say Sourceline(i)                     /* dump prolog                  */
 end;                                  /* 1 level, Do end.             */
$.@ps0=911                             /* set return code to exit      */
Return                                 /* r e t u r n                  */
/*--------------------------------------------------------------------*/
ERROR:                                 /* Routine: ERROR               */
If Arg() > 0 then rc=Arg(1)            /* get RC if directly passed    */
$.@ps0=rc                              /* error rc                     */
Say @.@sfn': Error occurred in line' SIGL', RC:'rc
Return                                 /* r e t u r n                  */
/********************************************************************/
Sub_groups:                            /* Routine: Sub_groups          */
Procedure expose @. $.                 /* procedure                    */
Numeric Digits 10                      /* current default is 9         */
Say @.@sfn':' Copies('-',70)           /* separator line               */
Say @.@sfn': POSIXGROUP  GroupID (GID)   Hex GID        Gaps (if any)'
Say @.@sfn':' Copies('-',70)           /* separator line               */
'PIPE STEM @.@direct.',                /* read the file                */
   '| LOCATE /POSIXGROUP/',            /* get these statements         */
   '| SPECS Words 2 1 Words 3 10-30 Right',
   '| SORT 10-30',                     /* sort the groups by GID       */
   '| STEM @.@pgroup.'                 /* output                       */
If @.@pgroup.0 = 0                     /* no POSIXGROUP statements?    */
 then Say @.@sfn': No POSIXGROUP statements found...'
 else Call Sub_groups1                 /* continue                     */
Return                                 /* r e t u r n                  */
/*-----------------------------------*/
Sub_groups1:                           /* Routine: Sub_groups1         */
```

```
ogid=-1                                /* initialize                  */
Do i=1 to @.@pgroup.0 while $.@ps0=0   /* 1 Do scan all posix groups  */
 Parse var @.@pgroup.i gname gid .     /* get group name and id       */
 If gid>0 then gid=Strip(gid,'L',0)    /* strip undesirable leading 0s */
 @.@GROUP.gid=gname                    /* save the name by id for later*/
 If gid <> ogid+1                      /* incorrect sequence?         */
  then Say @.@sfn':' Copies(' ',42) ogid+1'-'gid-1
 ogid=gid                              /* sequence is (now) correct   */
 Say @.@sfn':' Left(gname,10),         /* output group name           */
               Right(gid,12),          /* output group ID             */
               "    X'"||Right(D2X(gid),8,0)||"'" /* hex value of GID  */
 end;                                  /* 1 level, Do end.            */
Return                                 /* r e t u r n                 */
/*--------------------------------------------------------------------*/
Sub_users:                             /* Routine: Sub_users          */
Procedure expose @. $.                 /* procedure                   */
Say @.@sfn':' Copies('-',70)           /* separator line              */
Say @.@sfn': POSIXINFO       UID GNAME    SQE POSIXGLIST'
Say @.@sfn':' Copies('-',70)           /* separator line              */
'PIPE STEM @.@direct.',                /* read the file               */
   '| ZONE W 1 ALL /USER/ ! /POSIXGLIST/ ! /POSIXINFO/ ! /POSIXOPT/',
   '| NFIND *',                        /* avoid comment lines         */
   '| STEM @.@pusers. '                /* output                      */
Do i=1 to @.@pusers.0 while $.@ps0=0   /* 1 Do scan all user entries  */
 Parse var @.@pusers.i type .          /* get entry type              */
 Select                                /* 1 Select the directory opt. */
  when type='USER'       then Call Sub_users_User   @.@pusers.i
  when type='POSIXINFO'  then Call Sub_users_PInfo  @.@pusers.i
  when type='POSIXGLIST' then Call Sub_users_PGlist @.@pusers.i
  when type='POSIXOPT'   then Call Sub_users_POpt   @.@pusers.i
  otherwise Call ERROR 4747            /* card type not found         */
 end                                   /* 1 level, Select end.        */
end                                    /* 1 level, Do end.            */
Return                                 /* r e t u r n                 */
/*-----------------------------------*/
Sub_users_User:                        /* Routine: Sub_users_User     */
Procedure expose @. $.                 /* procedure                   */
If @.@PGlist='' & @.@PInfo='' & @.@POpt=''
 then Nop                              /* Not a POSIX user            */
 else Do                               /* 1 Do handle a line          */
       $.@Display=$.@Display+1         /* increment output line counter*/
       lc=$.@Display                   /* current output line count   */
       @.@DISPLAY.lc=@.@Sfn':' Left(@.@Userid,8),
                               Left(@.@PInfo,19),
                                    @.@POpt,
                                    @.@PGlist
       end;                            /* 1 level, Do end.           */
Parse Arg . @.@Userid .                /* get a new userid info       */
@.@PGlist=''                           /* init                        */
@.@PInfo=''                            /* init                        */
@.@POpt=''                             /* init                        */
Return                                 /* r e t u r n                 */
/*-----------------------------------*/
Sub_users_PGlist:                      /* Routine: Sub_users_PGlist   */
Procedure expose @. $.                 /* procedure                   */
Parse Arg 1 'GIDS' gids 'GNAMES' 1 'GNAMES ' gnames 73 .
If gids>0 then gids=Strip(gids,'L',0)  /* strip undesirable leading 0s */
gnames=Strip(gnames)                   /* get rid of extra blanks     */
If gids <> ''                          /* gids specified?             */
```

```
 then Do i=1 to Words(gids)            /* 1 Do translate into gnames   */
       gid=Word(gids,i)                /* handle gid's one by one      */
       gnames=gnames @.@GROUP.gid      /* translate it                 */
       end;                            /* 1 level, Do end.             */
@.@PGlist=gnames                       /* posix groups                 */
Return                                 /* r e t u r n                  */
/*-------------------------------------*/
Sub_users_PInfo:                       /* Routine: Sub_users_PInfo     */
Procedure expose @. $.                 /* procedure                    */
Parse Arg 1 'UID' uid . 1 'GID' gid . 1 'GNAME' gname .
If uid>0 then uid=Strip(uid,'L',0)     /* strip undesirable leading 0s */
If gid>0 then gid=Strip(gid,'L',0)     /* strip undesirable leading 0s */
If gname='' then gname=@.@GROUP.gid    /* translate it                 */
@.@PInfo=RIght(uid,10) Left(gname,8)   /* posix info                   */
Return                                 /* r e t u r n                  */
/*-------------------------------------*/
Sub_users_POpt:                        /* Routine: Sub_users_POpt      */
Procedure expose @. $.                 /* procedure                    */
Parse Arg 1 'SETIDS'      setids     .,/* setids                       */
          1 'QUERYDB'     querydb    .,/* querydb                      */
          1 'EXEC_SETIDS' execsetids . /* exec_setids                  */
@.@POpt=Translate(Left(setids,1),    '-',' ') ||,
        Translate(Left(querydb,1),   '-',' ') ||,
        Translate(Left(execsetids,1),'-',' ')
Return                                 /* r e t u r n                  */
```

B.3 BFSLIST Program Source Code

The following is the source for the four components of BFSLIST:

Figure 108 shows the source code for BFSLIST EXEC:

```
/* BFSLIST EXEC */
PARSE ARG PARMS'('OPTNS
IF PARMS='ROOT'| PARMS='root' then PARMS="/../VMBFS:VMSYS:ROOT/"
PARSE VALUE PARMS './' WITH PARMS .
IF RIGHT(PARMS,1) <> '/' THEN
   PARMS = PARMS'/'
RC = GET_DIR(PARMS)
IF RC = 0 THEN
   'XEDIT' USERID() 'BFSLIST A0 (PROFILE BFSLIST'
    DESBUF
EXIT RC

GET_DIR:
   PARSE ARG DIRPARM
   CURDIR = DIRPARM
   ADDRESS COMMAND
   'PIPE (NAME GETDIR END \)',
      '| BFSDIRECT' DIRPARM,
      '| SPEC $'DIRPARM'$ 1 1-* N',
      '| BFSSTATE',
      '| CHANGE 1.1 /-/F/',
      '| CHANGE 1.1 /D/D/',
      '| SPEC 1.1 10 2.9 NW 16.8 NW 25.13 NW 39.3 NW',
      '  43.4 NW $/$ N 47.2 N $/$ N 49.2 N 51.2 NW /:/ N',
      '  53.2 N /:/ N 55.2 N 58-* NW',
      '| STEM BFSDATA.'
RETURN RC
```

Figure 108. Source Code of BFSLIST EXEC

Figure 109 shows the source code for BFSLIST XEDIT:

```
/* BFSLIST XEDIT */
CALL PROLOG
EXIT

PROLOG:
   'PIPE VAR CURDIR 1 | VAR CURDIR'
   'EXTRACT /LSCREEN'
   'VERIFY 1' LSCREEN.2 - 1
   'SET RECFM V'
   'SET CURLINE ON 3'
   'SET MSGLINE ON -2'
   'SET TOFEOF OFF'
   'SET SCALE OFF'
   'SET PREFIX OFF'
   'SET RESERVED 2 BLUE NONE HI DIRECTORY:' CURDIR
   'SET RESERVED 3 GREEN NONE HI',
      ' CMD     T Privs     Owner    Group        Size',
      'Date       Time     FILEID'
   'SET CMDLINE BOTTOM'
   'SET RESERVED -4 BLUE NONE HI',
               '1= Help       2=          3= Quit   4= Delete     ',
               '5=             6= Sort(DATE)'
   'SET RESERVED -3 BLUE NONE HI',
               '7= Backward   8= Forward  9=       10=Dirlist     ',
               '11= XEDIT/List 12= Cursor'
   'SET SYN LINEND / SNAME CMS DMSXMS A 69 80 49 68/:1'
   'SET SYN LINEND / SSIZE CMS DMSXMS D 52 61  41 50/:1'
   'SET SYN LINEND / SDATE CMS DMSXMS D 49 68/:1'
   'SET SYN LINEND / REFRESHA COMMAND PRES/COMMAND',
      'MSGM OFF/COMMAND SCOPE A/:1 COMMAND DEL */COMMAND REST'
   'SET PF02 REFRESH'
   'SET PF14 REFRESH'
   'SET PF03 QQUIT'
   'SET PF15 QQUIT'
   'SET PF04 MACRO BFSDEL'
   'SET PF16 MACRO BFSDEL'
   'SET PF06 SSIZE'
   'SET PF18 SSIZE'
   'SET PF10 MACRO BFSEXAM 1'
   'SET PF22 MACRO BFSEXAM 1'
   'SET PF07 BACKWARD'
   'SET PF19 BACKWARD'
   'SET PF08 FORWARD'
   'SET PF20 FORWARD'
   'SET PF11 MACRO BFSEXAM 2'
   'SET PF22 MACRO BFSEXAM 2'
   CALL PUT_DIR
   'CURSOR FILE 1'
   'SET ALT 0'
RETURN

PUT_DIR:
   ':1'
   'SET EMSG OFF'
   'DELETE *'
   'SET EMSG ON'
   ':1'
   'PIPE (NAME PUTDIR)',
      '| STEM BFSDATA. 1',
      '| XEDIT'
   ':1'
RETURN
```

Figure 109. Source Code of BFSLIST XEDIT

Figure 110 shows the source code for BFSEXAM XEDIT:

```
/* BFSEXAM XEDIT */
Parse arg dirlist
address XEDIT
'EXTRACT /CURSOR'
'PIPE (name GETBFS)',
   '| stem BFSData. 1',
   '| spec w -1 1 w1 nw',
   '| stem BFSLine.'
'PIPE (name GETCUR)',
   '| var CurDir 1',
   '| var CurDir'
I_Line = Cursor.3
if I_Line > 0 & I_Line <= BFSLine.0 then
do
   parse var BFSLine.I_Line File Type
   select
      when Type = 'D' | Type = 'd' then
      do
        select
         when dirlist = '2' then 'EMSG The OBJECT is a Directory'
         when dirlist = '1' then
         'CMS BFSLIST' CurDir||File'/'
         if Rc <> 0 then
            'QQUIT' Rc
        otherwise
       end
      end
      when Type = 'F' | Type = 'f' then
        select
         when dirlist = '1' then 'EMSG The OBJECT is a File'
         when dirlist = '2' then
         address CMS "XEDIT '"CurDir||File"' (NAMETYPE BFS"
         if Rc <> 0 then
            'QQUIT' Rc
        otherwise
       end
      when Type = '?' | Type ='l' then
      do
        select
         when dirlist = '1' then do
          'CMS BFSLIST' CurDir||File'/'
           if Rc <> 0 then
             'QQUIT' Rc
         end
         when dirlist = '2' then do
         'PIPE (name GETLINK)',
            '| cms OPENVM QUERY LINK' CurDir||File'',
            '| drop first',
            '| spec w-3;4 1',
            "| change /'//",
            "| change /-/LINK/",
            '| var link'
            'EMSG The OBJECT is a ' link' use DIRLIST'
           end
        otherwise
        end
      end
    otherwise
   end
   'CURSOR FILE' Cursor.3
   'EXTRACT/LINE'
   ':'Cursor.3
   'COVERLAY *'
   ':'Line.1
   'SET ALT 0 0'
end
else
   'EMSG Cursor is not on valid data field'
```

Figure 110. Source Code of BFSEXAM XEDIT

Figure 111 shows the source code for BFSDEL XEDIT:

```
/* BFSDEL XEDIT */
Parse arg
address XEDIT
'EXTRACT /CURSOR'
'PIPE (name GETBFS)',
   '| stem BFSData. 1',
   '| spec w -1 1 w1 nw',
   '| stem BFSLine.'
'PIPE (name GETCUR)',
   '| var CurDir 1',
   '| var CurDir'
I_Line = Cursor.3
if I_Line > 0 & I_Line <= BFSLine.0 then
do
   parse var BFSLine.I_Line File Type
   select
      when Type = 'D' | Type = 'd' then
        do
         'EMSG You cannot DELETE a Directory'
         exit
        end
      when Type = 'F' | Type = 'f' then
       do
        Sure:
        say  'Are you sure you want a delete 'file' file ?'
        say  'Enter Yes or No'
        parse pull sino
        select
         when sino = 'N' | sino = 'n' | sino='no' | sino = 'NO' then
          do
           'VMFCLEAR'
           return
          end

     when sino = 'Y' | sino = 'y' | sino = 'yes' | sino = 'YES' then
          do
           'VMFCLEAR'
           'OPENVM ERASE ' Curdir||file
           return
          end
        otherwise do
                   say 'enter the right response, please'
                   call sure
                   return
                  end
       end
      when Type = '?' | Type ='l' then
        do
         'EMSG You cannot DELETE a LINK'
        end
    otherwise
   end
   'CURSOR FILE' Cursor.3
   'EXTRACT/LINE'
   ':'Cursor.3
   'COVERLAY *'
   ':'Line.1
   'SET ALT 0 0'
end
else
   'EMSG Cursor is not on valid data field'
```

Figure 111. Source Code of BFSDEL XEDIT

B.4 UNLOADBF EXEC

As noted during the BFS installation, if BFS errors occur and there is a need to regress, you would have to consult the LOADBFS control files and manually enter erase or delete commands, if appropriate. The UNLOADBF EXEC shown in Figure 112 reads a LOADBFS control file (backwards) and builds the necessary commands into an output file. This output file can then be reviewed by the invoker, and optionally run as required. A sample extract of the output file is shown in section 4.8.4, “LOADBFS Tables” on page 96, and in Figure 56 on page 98.

```
00001 /* +-------------------------------------------------------------------+
00002    | Format: unloadbf fn <ft|LOADBFS> <fm|*>                           |
00003    | Purpose:         Remove BFS files using a LOADBFS control file    |
00004    |                  as input.  Will read the file backwards and      |
00005    |                  look for the appropriate keywords to perform     |
00006    |                  the OPENVM MOUNT, OPENVM ERASE, and DELETE USER. |
00007    | Output:          Creates file fn UNLOADBF A. Check this file and  |
00008    |                  change as required.  Rename or EXECLOAD as EXEC  |
00009    |                  and run as required.                             |
00010    +---------------------------------------------------- SG24-4747 +
00011 */
00012 Address "CMS"
00013 Arg fn ft fm  .
00014 If fn=""|fn="?"|fn="HELP" Then Do       /* Show prologue (help)   */
00015    Parse Source . . fn ft .
00016    "pipe <" fn ft "* |",
00017         "tolabel */|",
00018         "specs 3-* 1 |",
00019         "cons"
00020    Exit -3                              /* Exit after help shown  */
00021 End
00022 If fm="" Then fm="*"                    /* Set defaults           */
00022 If ft="" Then ft="LOADBFS"
00023 /* +-----------------------------------------------------------+
00024    | This next pipe will create a prolog for out output file. |
00025    +-----------------------------------------------------------+   */
00026 "pipe literal Address 'CMS' |",
00027 "literal /*-------------------------------- SG24-4747 */|",
00028 "literal /*             Rundate:" Right(Date(),11),
00039                                   Left(Time(),5) "    */  |",
00030 "literal /* order to run these commands.                */  |",
00031 "literal /* Note: Must have proper permissions in      */  |",
00032 "literal /* this mount as required.                     */  |",
00033 "literal /* bottom of this file and move or execute    */  |",
00034 "literal /* If there is a 'MOUNT' required, check the  */  |",
00035 "literal /* Change as required and run as an EXEC.     */  |",
00036 "literal /* Output from execution of UNLOADBF" Left(Strip(fn),8) "*/ |",
00037 "literal /*-----------------------------------------*/|",
00038 ">" fn "UNLOADBF A"
```

Figure 112. (Part 1 of 2) UNLOADBF EXEC

```
00039 /* +-----------------------------------------------------------+
00040    | The main pipe which will select and build our output file |
00041    +---------------------------------------------- SG24-4747 +  */
00042 Signal On Error
00043 "pipe (endchar ?) fileback" fn ft fm"|",/* Read file backwards    */
00044    "mount: nfind MOUNT |",              /* Send MOUNT to mount:   */
00045    "enroll: nfind ENROLL |",            /* Send ENROLL to enroll: */
00046    "all /EXTLINK/ ! /MKNODE/",          /* Use 'all' stage to ... */
00047       "! /SLINK/ ! /FILE/",             /* ... select only the    */
00048       "! /DIR/ |",                      /* ... records we need.   */
00049    "specs /'OPENVM ERASE / 1 w2 n",     /* Build command string...*/
00050       "/'/ n |",                        /* ...                    */
00051    "fi: fanin |",                       /* Get mount and enroll   */
00052    "all ·(/syntax:/ ! /MKNODE/ !",      /* Remove unwanted hitch- */
00053       " /SLINK/ ! /DIR/ ! /FILE/ !",    /* hikers ...             */
00054       " /EXTLINK/ ! /ENROLL/) |",       /* ...                    */
00055    ">>" fn "UNLOADBF A",                /* Write to output file   */
00056    "?mount: |",
00057    "specs /'OPENVM MOUNT / 1 w2-* n",   /* Build command string...*/
00058       "/'/ n |",                        /* ...                    */
00059    "buffer|",
00060    "fi:",                               /* Connect to fanin       */
00061    "?enroll: |",
00062    "specs /'DELETE USER / 1",           /* Build command string...*/
00063    " w2 n w7 nw /'/ n|",                /* ...                    */
00064    "buffer|",
00065    "fi:"                                /* Connect to fanin       */
00066 Say "File:" fn "UNLOADBF A has been created.  Please review and"
00067 Say "      follow instructions in this file."
00068 Exit
00069 Error:
00070   Say "Error" rc "on line" sigl"."
00071   Say Sourceline(sigl)
00072   Exit rc
```

Figure 113. (Part 2 of 2) UNLOADBF EXEC

B.5 DU Program Source Code

Figure 114 on page 198 contains the contents of the DU EXEC.

```
/* ------------------------------------------------------------- */
/*  DU EXEC                                                      */
/*                                                               */
/* List the TOTAL of BLOCKS of the files in a Directory          */
/*                                                               */
/*  COMPONENTS     DU EXEC                                       */
/*                 DU REXX                                       */
/* ------------------------------------------------------------- */

PARSE ARG PARMS'('OPTNS
IF PARMS='ROOT'|  PARMS='root' then PARMS="/../VMBFS:VMSYS:ROOT/"
PARSE VALUE PARMS './' WITH PARMS .

GET_DU:
IF RIGHT(PARMS,1) <> '/' THEN
   PARMS = PARMS'/'
   CURDIR = PARMS
   ADDRESS COMMAND
   'PIPE (NAME GETDIR END ?)',
      '| BFSDIRECT' PARMS,
      '| SPEC $'PARMS'$ 1 1-* N',
      '| BFSSTATE',
      '| a: fanout',
      '| elastic',
      '| b: SPEC 58-* 12',
           'select 1 1-40 1 ',
      '| CONSOLE ',
      '?',
      'a:',
      '| rexx du',
      '| b:'
RETURN RC
```

Figure 114. DU EXEC Source

Figure 115 on page 199 contains the source of the DU REXX.

```
/* ------------------------------------------------------------- */
/*  DU REXX                                                      */
/*                                                               */
/* List the TOTAL of BLOCKS of the files in a Directory          */
/*                                                               */
/*  COMPONENTS     DU EXEC                                       */
/*                 DU REXX                                       */
/* ------------------------------------------------------------- */

sum_blocks=0
PARSE ARG PARMS
 'CALLPIPE',
      '*.input:',
      '| SPEC 39.3 10 ',
      '| stem blocks.'
do i=1 to blocks.0
   f=1
   if blocks.i = 0 then f=0
   blocks.i= (blocks.i%512)+f
   sum_blocks=(blocks.i+sum_blocks)
   'output ' blocks.i '     ./'
end
'output'
'output' sum_blocks '  BLOCKS TOTAL (512 Bytes per Block)'
RETURN RC
```

Figure 115. DU REXX Source

Appendix C. Special Notices

This publication is written to assist technical professionals who wish to learn more about the POSIX implementation on VM. The information in this publication is not intended as the specification of any programming interfaces that are provided by VM/ESA Version 2 Release 1.0. See the PUBLICATIONS section of the IBM Programming Announcement for VM/ESA Version 2 Release 1.0 for more information about what publications are considered to be product documentation.

References in this publication to IBM products, programs or services do not imply that IBM intends to make these available in all countries in which IBM operates. Any reference to an IBM product, program, or service is not intended to state or imply that only IBM′s product, program, or service may be used. Any functionally equivalent program that does not infringe any of IBM′s intellectual property rights may be used instead of the IBM product, program or service.

Information in this book was developed in conjunction with use of the equipment specified, and is limited in application to those specific hardware and software products and levels.

IBM may have patents or pending patent applications covering subject matter in this document. The furnishing of this document does not give you any license to these patents. You can send license inquiries, in writing, to the IBM Director of Licensing, IBM Corporation, 500 Columbus Avenue, Thornwood, NY 10594 USA.

The information contained in this document has not been submitted to any formal IBM test and is distributed AS IS. The information about non-IBM (″vendor″) products in this manual has been supplied by the vendor and IBM assumes no responsibility for its accuracy or completeness. The use of this information or the implementation of any of these techniques is a customer responsibility and depends on the customer′s ability to evaluate and integrate them into the customer′s operational environment. While each item may have been reviewed by IBM for accuracy in a specific situation, there is no guarantee that the same or similar results will be obtained elsewhere. Customers attempting to adapt these techniques to their own environments do so at their own risk.

Reference to PTF numbers that have not been released through the normal distribution process does not imply general availability. The purpose of including these reference numbers is to alert IBM customers to specific information relative to the implementation of the PTF when it becomes available to each customer according to the normal IBM PTF distribution process.

The following terms are trademarks of the International Business Machines Corporation in the United States and/or other countries:

Advanced Function Printing	AFP
AIX	C/VM
C/370	Current
DFSMS	DFSMS/VM
ES/9000	IBM
Language Environment	MVS
OpenEdition	OS/2
RACF	S/390

System/390	Virtual Machine/Enterprise Systems Architecture
VM/ESA	VTAM

The following terms are trademarks of other companies:

C-bus is a trademark of Corollary, Inc.

PC Direct is a trademark of Ziff Communications Company and is used by IBM Corporation under license.

UNIX is a registered trademark in the United States and other countries licensed exclusively through X/Open Company Limited.

Microsoft, Windows, and the Windows 95 logo are trademarks or registered trademarks of Microsoft Corporation.

AT&T	American Telephone and Telegraph Company
Bell	American Telephone and Telegraph Company
DCE	The Open Software Foundation
Open Software Foundation	The Open Software Foundation
OSF	The Open Software Foundation
POSIX	Institute of Electrical and Electronic Engineers
UNIX System V	AT&T Bell Laboratories Incorporated
X/Open	X/Open Company Limited

Other trademarks are trademarks of their respective companies.

Appendix D. Related Publications

The publications listed in this section are considered particularly suitable for a more detailed discussion of the topics covered in this redbook.

D.1 International Technical Support Organization Publications

For information on ordering these ITSO publications, see “How To Get ITSO Redbooks” on page 205.

- *VM/ESA OpenEdition DCE Introduction and Implementation Notebook*, SG24-4554
- *Plumbers' Workbench*, SG24-4523
- *GUI Facility Developer's Guide Using C++*, SG24-2542
- *Elements of Security: AIX 4.1*, GG24-4433
- *Client/Server Computing with VM/ESA as Part of the Open Enterprise*, GG24-3950
- *VM/ESA Version 2 Overview and Technical Guide*, GG24-4418

A complete list of International Technical Support Organization publications, known as redbooks, with a brief description of each, may be found in:

International Technical Support Organization Bibliography of Redbooks, GG24-3070.

D.2 Other IBM Publications

These publications are also relevant as further information sources.

- *VM/ESA: CMS Command Reference*, SC24-5776
- *VM/ESA: CMS Application Development Guide*, SC24-5761
- *VM/ESA: CMS Application Development Reference*, SC24-5762
- *VM/ESA: CMS File Pool Planning, Administration, and Operation*, SC24-5751
- *VM/ESA: VM/SES Introduction and Reference*, SC24-5747
- *VM/ESA: Service Guide*, SC24-5749
- *IBM OpenEdition for VM/ESA: POSIX Conformance Document*, SC24-5725
- *IBM OpenEdition for VM/ESA: Callable Services Reference*, SC24-5726
- *IBM OpenEdition for VM/ESA: User's Guide*, SC24-5727
- *IBM OpenEdition for VM/ESA: Command Reference*, SC24-5728
- *IBM OpenEdition for VM/ESA: Advanced Application Programming Tools*, SC24-5729
- *IBM OpenEdition for VM/ESA: Sockets Reference*, SC24-5741
- *TCP/IP for VM/ESA Messages and Codes*, SC31-6151
- *TCP/IP for VM/ESA User's Guide*, SC31-6081
- *TCP/IP for VM/ESA Planning and Customization Guide*, SC31-6082

- *IBM C for VM/ESA: Licensed Program Specifications*, GC09-2148
- *IBM C for VM/ESA: User's Guide*, SC09-2152
- *IBM C for VM/ESA: Programming Guide*, SC09-1384
- *IBM C for VM/ESA: Language Reference*, SC09-2153
- *IBM C for VM/ESA: Library Reference*, SC23-3908
- *IBM C for VM/ESA: Diagnosis Guide*, SC09-2149
- *IBM C for VM/ESA: Compiler and Run-Time Migration Guide*, SC09-2147
- *IBM Language Environment for MVS & VM Fact Sheet*, GC26-4785
- *IBM Language Environment for MVS & VM Concepts Guide*, GC26-4786
- *IBM Language Environment for MVS & VM Installation and Customization Guide*, SC26-4817
- *IBM Language Environment for MVS & VM Programming Guide and Reference*, SC26-4818
- *IBM Language Environment for MVS & VM Programming Reference*, SC26-3312
- *IBM Language Environment for MVS & VM Licensed Program Specifications*, GC26-4774

D.3 On Your Local Bookstand

These publications are available through your local book vendor, or through IBM.

- Rosenblatt, Bill. 1993. *Learning the Korn Shell*, SR28-5268 (ISBN 1-56592-054-6). O'Reilly & Associates, Inc.
- Morris I. Bolsky and David G. Korn. 1995. *The New Korn Shell*, SR28-5706 (ISBN 0-13-182700-6). New Jersey: Prentice Hall PTR.

D.4 Getting the Redbook Sample Programs from the Net

The sample programs contained in this publication are available through the Internet and the IBM VNET network.

Internet: If you have an Internet connection, you can access the program materials by anonymous FTP:

```
ftp ftp.almaden.ibm.com
user: anonymous
password: your E-mail address
dir
cd redbooks/SG244747
binary
mget *.*
quit
```

README.1ST, part of the zip file, contains additional information.

VNET: Users of the IBM VNET network may get the SG244747 PACKAGE, which is stored in the VMTOOLS repository. Issue the following CMS command:

```
TOOLS TO VMTOOLS GET SG244747 PACKAGE
```

How To Get ITSO Redbooks

This section explains how both customers and IBM employees can find out about ITSO redbooks, CD-ROMs, workshops, and residencies. A form for ordering books and CD-ROMs is also provided.

This information was current at the time of publication, but is continually subject to change. The latest information may be found at URL http://www.redbooks.ibm.com/redbooks.

How IBM Employees Can Get ITSO Redbooks

Employees may request ITSO deliverables (redbooks, BookManager BOOKs, and CD-ROMs) and information about redbooks, workshops, and residencies in the following ways:

- **PUBORDER** — to order hardcopies in United States
- **GOPHER link to the Internet** - type GOPHER.WTSCPOK.ITSO.IBM.COM
- **Tools disks**

  To get LIST3820s of redbooks, type one of the following commands:

  ```
  TOOLS SENDTO EHONE4 TOOLS2 REDPRINT GET SG24xxxx PACKAGE
  TOOLS SENDTO CANVM2 TOOLS REDPRINT GET SG24xxxx PACKAGE (Canadian users only)
  ```

  To get lists of redbooks:

  ```
  TOOLS SENDTO WTSCPOK TOOLS REDBOOKS GET REDBOOKS CATALOG
  TOOLS SENDTO USDIST MKTTOOLS MKTTOOLS GET ITSOCAT TXT
  TOOLS SENDTO USDIST MKTTOOLS MKTTOOLS GET LISTSERV PACKAGE
  ```

  To register for information on workshops, residencies, and redbooks:

  ```
  TOOLS SENDTO WTSCPOK TOOLS ZDISK GET ITSOREGI 1996
  ```

  For a list of product area specialists in the ITSO:

  ```
  TOOLS SENDTO WTSCPOK TOOLS ZDISK GET ORGCARD PACKAGE
  ```

- **Redbooks Home Page on the World Wide Web**

  http://w3.itso.ibm.com/redbooks/redbooks.html

- **IBM Direct Publications Catalog on the World Wide Web**

  http://www.elink.ibmlink.ibm.com/pbl/pbl

  IBM employees may obtain LIST3820s of redbooks from this page.

- **ITSO4USA category on INEWS**
- **Online** — send orders to: USIB6FPL at IBMMAIL or DKIBMBSH at IBMMAIL
- **Internet Listserver**

  With an Internet E-mail address, anyone can subscribe to an IBM Announcement Listserver. To initiate the service, send an E-mail note to announce@webster.ibmlink.ibm.com with the keyword subscribe in the body of the note (leave the subject line blank). A category form and detailed instructions will be sent to you.

How Customers Can Get ITSO Redbooks

Customers may request ITSO deliverables (redbooks, BookManager BOOKs, and CD-ROMs) and information about redbooks, workshops, and residencies in the following ways:

- **Online Orders** (Do not send credit card information over the Internet) — send orders to:

	IBMMAIL	**Internet**
In United States:	usib6fpl at ibmmail	usib6fpl@ibmmail.com
In Canada:	caibmbkz at ibmmail	lmannix@vnet.ibm.com
Outside North America:	bookshop at dkibmbsh at ibmmail	bookshop@dk.ibm.com

- **Telephone orders**

United States (toll free)	1-800-879-2755
Canada (toll free)	1-800-IBM-4YOU
Outside North America	(long distance charges apply)
(+45) 4810-1320 - Danish	(+45) 4810-1020 - German
(+45) 4810-1420 - Dutch	(+45) 4810-1620 - Italian
(+45) 4810-1540 - English	(+45) 4810-1270 - Norwegian
(+45) 4810-1670 - Finnish	(+45) 4810-1120 - Spanish
(+45) 4810-1220 - French	(+45) 4810-1170 - Swedish

- **Mail Orders** — send orders to:

IBM Publications	IBM Publications	IBM Direct Services
Publications Customer Support	144-4th Avenue, S.W.	Sortemosevej 21
P.O. Box 29554	Calgary, Alberta T2P 3N5	DK-3450 Allerød
Raleigh, NC 27626-0570	Canada	Denmark
USA		

- **Fax** — send orders to:

United States (toll free)	1-800-445-9269
Canada (toll free)	1-800-267-4455
Outside North America	(+45) 48 14 2207 (long distance charge)

- **1-800-IBM-4FAX (United States)** or **(+1) 415 855 43 29 (Outside USA)** — ask for:

  Index # 4421 Abstracts of new redbooks
  Index # 4422 IBM redbooks
  Index # 4420 Redbooks for last six months

- **Direct Services** - send note to softwareshop@vnet.ibm.com
- **On the World Wide Web**

Redbooks Home Page	http://www.redbooks.ibm.com/redbooks
IBM Direct Publications Catalog	http://www.elink.ibmlink.ibm.com/pbl/pbl

- **Internet Listserver**

  With an Internet E-mail address, anyone can subscribe to an IBM Announcement Listserver. To initiate the service, send an E-mail note to announce@webster.ibmlink.ibm.com with the keyword subscribe in the body of the note (leave the subject line blank).

IBM Redbook Order Form

Please send me the following:

Title	Order Number	Quantity

- **Please put me on the mailing list for updated versions of the IBM Redbook Catalog.**

First name Last name

Company

Address

City Postal code Country

Telephone number Telefax number VAT number

- Invoice to customer number _____
- Credit card number _____

Credit card expiration date Card issued to Signature

We accept American Express, Diners, Eurocard, Master Card, and Visa. Payment by credit card not available in all countries. Signature mandatory for credit card payment.

DO NOT SEND CREDIT CARD INFORMATION OVER THE INTERNET.

Glossary

A

access permission. A group of designations that determine who can access a particular file and how the user can access the file.

alias. An alternate name for a shell command. An alias for a command can be defined with options different from those of the command itself. An alias can be a convenient shorthand for a complicated command line, such as a pipeline.

archive. Synonymous with backup, and backup copy.

awk. A file processing language that is well-suited to data manipulation and retrieval of information from text files. **awk** is named for its creators, Alfred V. Aho, Peter J. Weinberger, and Brian W. Kernighan.

B

background. In multiprogramming, the conditions under which low-priority programs are processed.

backslash. The character "\"—also known as a *reverse solidus*. The backslash enables a user to escape the special meaning of a character. That is, typing a backslash before a character tells the system to ignore any special meaning the character might have.

basename. The final, or only, file name in a path name.

BFS. Byte File System. (BFS) server of VM/ESA.

Byte File System. A file system where a file consists of an ordered sequence of bytes, rather than records as in the CMS file system. BFS files may be organized into hierarchical directories.

C

canonical input. If a terminal is in canonical mode, input is collected and processed one line at a time. A read request is not returned until the line is terminated or a signal is received. Erase and kill processing is performed on input that was not terminated by one of the line termination characters. Erase processing removes the last character in the line, and kill processing removes the entire line.

character special file. A special BFS file that provides access to an input or output device. The character interface is used for devices that do not use block I/O.

code page. A table showing codes assigned to character sets.

CMS short name. The CMS short name is a numerical identifier unique to every file in the BFS. See also **inode**.

command alias. An abbreviation of a long command line or a new name for a command.

command interpreter. A program that reads the commands that you type and then processes them. When you are typing commands into the computer, you are actually typing input to the command interpreter. The interpreter then decides how to perform the commands that you have typed. The shell is an example of a command interpreter. Synonymous with command language interpreter.

current directory. The directory that is active and can be displayed with the **pwd** command. Synonymous with current working directory.

current working directory. The BFS directory a user is working in.

D

default directory. The directory name supplied by the operating system if a name is not specified.

directory. A binary file consisting of a list of other files.

dump. To write at a particular instant the contents of storage, or a part of storage, onto another data medium for the purpose of safeguarding or debugging the data.

E

EBCDIC. Extended binary-coded decimal interchange code.

effective user ID. The current user ID, but not necessarily the user's login ID. For example, a user logged in under a login ID may change to another user's ID. This ID becomes the effective user ID until the user switches back to the original login ID.

external link. A file system entry that can be used to:

- Reference data outside of BFS (data residing on a CMS minidisk or in a directory)
- Create an implicit mount point
- Contain data in an application-defined format

F

FIFO special file. A type of file with the property that data written to the file is read on a first-in-first-out basis.

file mode. File mode on a UNIX system; refers to file protection access of a file. There are three types of file access: read, write, and execute, designated by the letters `r`, `w`, and `x` respectively.

file pool. A file pool is a collection of minidisks owned by a particular file pool virtual machine known as a file pool server. The minidisks are used for storing file pool repository data and control data.

file space. Storage allocated to an enrolled user within the Shared File System (SFS) or Byte File System (BFS). In BFS terms, a file space can also be mounted as a file system.

file type. On UNIX systems, it is safe to say that everything is a file. The operating system even represents I/O devices as files. There are several different kinds of files, each with a different purpose.

Regular file - Files containing data. These may be text files, data files, executable files, and so on.

Special files - Used for device I/O under UNIX. They reside in directory `/dev` and its subdirectories.

Link - Allows several file names to refer to a single file on disk. There are two types of links, *hard links* or *soft links*. A *hard link* uses the inode address of a file, while a *soft link* will use a file's directory.

foreground process. A process that is a member of a foreground process group. Typically, this is the application the user interacts with.

function key. A key that causes a specified sequence of operations to be performed when it is pressed. Generally used to refer to keys labeled <F*n*>, for example <F1>.

G

GID. An acronym for group identifier.

group ID. A unique number assigned to a group of related users. The GID can often be substituted in commands that take a group name as an operand.

group name. A name that uniquely identifies a group of POSIX users to the system.

H

history file. A file that lists the shell commands that have been processed.

home directory. The current directory associated with the user at the time of login.

I

inode. A data structure on disk that describes and stores a file's attributes, including:

- UID and GID of the file
- File type (regular, directory, and so on)
- Access modes (permissions)
- File creation, modification, and access times
- Inode modification time
- Number of links to the file
- Size of the file

It is possible for two files to have the same inode number when hard links are used for files.

K

kernel. The part of the OpenEdition for VM/ESA component containing programs for such tasks as I/O, management, and communication.

kill. An operating system command that stops a process.

L

line mode. Synonym for *canonical mode*.

locale. The definition of the subset of a user's environment, which depends on language and cultural conventions.

login. In UNIX systems, the act of gaining access to a computer system by entering identification and authentication information at the workstation.

M

mode. A collection of attributes that specifies a file's type and its access permissions.

mount. To make a file system accessible.

mount point. The path name of the directory on which the file system is mounted.

N

noncanonical mode. An input processing mode where input character erase and killing are eliminated, making input characters available to the user program as they are typed.

O

output file. A file that a program opens so that it can write to that file.

output redirection. The specification of an output destination other than the standard one.

owner. The user who has the highest level of access authority to a data object or action, as defined by the object or action.

P

parent directory. The directory that both contains a directory entry for the given director and is represented by the path name .. in the given directory.

path name. A BFS file name specifying all BFS directories leading to the file.

permission mode. A three-digit octal code or a nine-letter alphabetic code that determines how a file can be used. It indicates the read, write, and run permissions for the owner, for the group, and for all others.

pipe. An interprocess communication mechanism that connects an output file descriptor to an input file descriptor. Usually the standard output of one process is connected to the standard input of another, forming a pipeline.

portable character set. The set of characters described in POSIX.2 2.4 that is supported on all conforming systems.

POSIX. Portable Operating System Interface for Computer Environments; an IEEE operating system standard, closely related to the UNIX system (software writing).

process. A function being performed or waiting to be performed.

prompt. A displayed symbol or message that requests information or operator action.

R

RACF. Resource Access Control Facility, an IBM security product.

relative path name. The name of a BFS directory or file expressed as a sequence of directories followed by a file name, beginning from the current directory.

root. The starting point of the Byte File System. It can also be the user name for the system user with the most authority.

root directory. The BFS directory that contains all other directories in the system.

S

session. The period of time during which a user of a terminal can communicate with an interactive system. It is usually the elapsed time between logon and logoff.

set-group-ID mode bit. In setting file access permissions, the bit that sets the effective group ID of the process to the file′s group upon processing.

set-user-ID mode bit. In setting file access permissions, the bit that sets the effective user ID of the process to the file′s owner upon processing.

shell. (1) A program that interprets and processes interactive commands from a terminal or file. (2) An interface between the user and the operating system. (3) The working environment for an interactive user.

shell script. A file of shell commands very similar to a CMS EXEC.

socket. A method of communication between two processes. Sockets allow communication in two directions, in contrast to pipes, which allow communication in only one direction. The processes using a socket can be on the same system or on systems in the same network.

sticky bit. A permission bit that causes an executable program to remain in storage after execution. *Sticky bit* is the common name of save text mode. The sticky bit can be set, but VM/ESA will take no action based on the setting.

storage group. A collection of minidisks within a file pool. Each storage group is identified by a number. Storage group 1 is for catalog information only. Storage groups 2 through 32767 are for user data.

superuser. A privileged account with unlimited access to all files and commands. Many administrative tasks require superuser status.

symbolic link. A file which points to another path name in the file system. Symbolic links may span VM/ESA BFS file pools.

U

UID. An acronym for user identification. See **user ID**.

user ID. A unique string of characters that identifies a user to the system. This string of characters limits the functions and information the user can use.

W

white space. Space characters, tab characters, new line characters, and comments. White space separates commands on the command line.

working directory. The active directory used to resolve path names that do not begin with a slash. In similar systems, a working directory may be called the current directory or the current working directory.

List of Abbreviations

ABEND	abnormal end
ACCT	user accounting information data set
ADMIN	administrative/administration
AIX	advanced interactive executive (IBM's UNIX)
APAR	authorized program analysis report
API	application program interface
APPC/VM	advanced program-to-program communication/virtual machine (IBM)
ASCII	American National Standard Code for Information Interchange
AUX	auxiliary
AVS	APPC/VM VTAM support
BFS	Byte File System
BSD	Berkeley software distribution (UC at Berkeley, UNIX)
CMD	command
CMS	conversational monitor system (VM-based software, IBM)
COBOL	common business oriented language
CON	console
CONFIG	configuration/configure
CP	control program
CPU	central processing unit
CRR	coordinated resource recovery (VM/ESA)
CSL	callable services library (VM/CMS HLL-callable services)
DASD	direct access storage device
DCE	Distributed Computing Environment (OSF)
DCSS	discontiguous saved segment (VM/ESA)
DFSMS	Data Facility Storage Management Subsystem (MVS and VM)
DIR	directory
DISC	disconnect
DSECT	dummy control section
EBCDIC	extended binary coded decimal interchange code
ESM	external security manager (VM/CMS SFS authorization exits)
ESP	early support program (software)
EXEC	execution program
FTP	file transfer program
GB	gigabyte (10**9 bytes or 1,000,000,000 bytes)
GCS	group control system (VM/VTAM)
GID	group identifier
GUI	graphical user interface
HX	halt execution
I/O	input/output
IEEE	Institute of Electrical and Electronics Engineers
IPL	initial program load
ISMF	interactive storage management facility (IBM software product)
IT	information technology
ITSO	International Technical Support Organization
IUCV	inter-user communication vehicle
KB	kilobyte, 1000 bytes (1024 bytes memory) case should be Kb
LRECL	logical record length
MACH	machine
MB	megabyte, 1,000,000 bytes (1,048,576 bytes memory) case should be Mb
MVS	multiple virtual storage (IBM System 370 & 390)
OE	Open Edition
OS	operating system
OSF	Open Software Foundation Inc. (IBM with DEC, Apollo, HP, Groupe Bull, Nixdorf, Siemens)

POSIX	portable operating system interface for computer environments, an IEEE operating system standard, closely related to the UNIX system (software writing)
PPF	program product file
PRT	print
PTF	program temporary fix
PUN	punch
PWD	print working directory (AIX)
RACF	resource access control facility
RDR	reader
RECFM	record format
RECS	records
REL	relative
REXX	restructured extended executor language
RSCS	remote spooling communications subsystem (VM′s counterpart to MVS JES NJE)
RTL	run time library
SCIF	single console image facility
SFS	Shared File System (hierarchical sharable VM/CMS file system)
SMS	system managed storage (MVS and VM)
SU	switch user (AIX)
SYN	synonym
TCP/IP	Transmission Control Protocol/Internet Protocol (USA, DoD, ARPANET; TCP=layer 4, IP=layer 3, UNIX-ish/Ethernet-based system-interconnect protocol)
TELNET	U.S. Dept. of Defense′s virtual terminal protocol, based on TCP/IP
TMP	temporary
TSAF	transparent services access facility
UID	user identification
UNIX	an operating system developed at Bell Laboratories (trademark of UNIX System Laboratories, licensed exclusively by X/Open Company, Ltd.)
URL	Uniform Resource Locator
UTC	universal time, coordinated
VM	virtual machine (IBM System 370 & 390)
VM/ESA	virtual machine/enterprise systems architecture (IBM)
VMLIB	VM library (system supplied VM/CMS CSL routines)
VMSES	virtual machine serviceability enhancements staged
VSE	virtual storage extended (IBM System/370)
VTAM	virtual telecommunications access method (IBM) (runs under MVS, VM, & DOS/VSE)
XEDIT	extended editor (the IBM VM system editor that allows text input & revision)

Index

Special Characters

- character
  - environment variable 28
  - portable character 11
_, portable character 11
? character
  - portable character 11
  - shell variable 171
/, portable character 11
/home, a BFS directory 10
/tmp directory 35
/tmp, BFS install 89
/var, BFS install 89
., portable character 11
.profile
  - shell 22, 107
.sh_history (shell history file) 24, 111
.SHOW LOCK USER, SFS command 115, 116
.SHOW LOCK WANTLOCK, SFS command 115, 116
″, portable character 11
(, portable character 11
), portable character 11
{, portable character 11
}, portable character 11
@, positional parameter shell 169
¢, escape character 30
$ 112
  - portable character 11
  - shell parameter 170
  - shell prompt 138
$ character, value of variables 169
$-, shell parameter 170
$?, shell parameter 170
$@, shell parameter 169, 170
$* characters
  - shell parameter 169, 170
$1 characters
  - shell argument 169, 170
$N, shell argument number 170
$PPF 134, 135
* character
  - portable character 11
  - shell parameter 169
\, portable character 11
& character
  - portable character 11
  - shell programming 178
#, portable character 11
#, shell positional parameter 170
#, shell prompt 112, 138
##, matching pattern 171
%, matching pattern 171
%%, matching pattern 171
<, portable character 11
<limits.h>, C header file 8
>, portable character 11
| character
  - portable character 11
  - shell pipe separator 176, 178
||, shell programming 178

Numerics

1003.1, POSIX interface 133
1003.1a, POSIX interface 133
1003.1c, POSIX interface 133
132 columns displays 34
3270
  - display device 29, 30

A

abbreviations 213
absolute path name 38
accept command 44
accessing files 15
ACCESSLIST, BFS server 85
acct (FTP command) 54
ACCT, file pool option 84
acronyms 213
ADDRESPACE, BFS server 85
ADDRESS OPENVM statement 150
ADDUSER, tool to create home directory′s 153
adm, default user 143
ADMIN, file pool control statement 86
administration
  - customizing shell startup 93
  - deleting a user 157
  - enrolling new users 94
  - file pool machine 77
  - file space limits 112
  - locating system dumps 128
  - machine, regarding BFS 85
  - moving data to the BFS 50
  - OPENVM commands 50
  - pax backups 50
  - sample shell script for XEDIT 38
  - SFS, SFSDOT LIST3270 115
  - system 107
  - UID and GID 139
  - user 137
Advanced Function Printing 45
advantages and disadvantages, dedicated BFS server 90
alias
  - creating for help 26
  - defined 209
  - example 109

alias *(continued)*
history (shell command) 24
r command 25
shell command 109
x for XEDIT 24
allocations
initial DASD 81
MAXCONN 81
ALLOW, POSIXOPT SETIDS 85
amount of paging 35
ANSI-C 8
API, overview 132
APPC, library 134
APPC/VM VTAM Support 132
append (FTP command) 54
application code, POSIX, saved segment 129
application development 6
application portability
POSIX 1, 164
application program interface 132
archive
creating 50
defined 209
files 65
pax 59
tar 57
arrays
in shell scripts 172
variables, shell programming 168
ascii (FTP command) 55
assembler, as used in UNIX 5
assigning
UIDs, planning 72
VM users to POSIX user groups 73
attributes
shell 173
shell programming 168
authorized group 146
authors, document xiv
AUTOLOGON, file pool setup 88
AVS 135
AVS, remote BFS access 132
awk (shell command) 209

B

background, defined 209
backslash, defined 209
backspace character, control-H 31
backup
BFS backup
concurrent 116
considerations 114
methods 113, 114
restoring individual files 119
cpio (using) 58
creating 50
file system
.SHOW LOCK USER command 115
.SHOW LOCK WANTLOCK command 115
backup *(continued)*
file system *(continued)*
CMS short name 120
concurrent 116
considerations 114
FILEPOOL BACKUP command 114, 116
FILEPOOL FILELOAD command 114, 119
FILEPOOL LIST BACKUP command 114, 119
FILEPOOL RELOAD command 114
FILEPOOL RELOAD FILES 120
FILEPOOL RELOAD FILES command 119
FILEPOOL UNLOAD command 114, 116
inodes 120
locks 114
methods 113, 114
QUERY FILEPOOL CONFLICT command 115
restoring individual files 119
SET FILEWAIT command 115
SFSDOT LIST3270 115
pax (using) 59
tar (using) 57
using pax 50
BACKUP, file pool control statement 86
base $PPF file name 135
basename, defined 209
basic constructs, shell scripts 167
Bell labs 5
Berkeley, University of (UNIX introduction) 5
BFS
.profile system profile 107
backout tool 196
backup
.SHOW FILEPOOL USER command 115
.SHOW LOCK WANTLOCK command 115
CMS short name 120
concurrent 116
considerations 114
FILEPOOL BACKUP command 114, 116
FILEPOOL FILELOAD command 114, 119
FILEPOOL LIST BACKUP command 114
FILEPOOL LISTBACKUP command 119
FILEPOOL RELOAD command 114
FILEPOOL RELOAD FILES 120
FILEPOOL RELOAD FILES command 119
FILEPOOL UNLOAD command 114, 116
inodes 120
locks 114
methods 113, 114
QUERY FILEPOOL CONFLICT command 115
restoring individual files 119
SET FILEWAIT command 115
SFSDOT LIST3270 115
BFSLIST 181
BFSROOT installation tool 79
counting blocks of data 69
creating backup with pax 50
dedicated server 80
default file spaces 80

BFS *(continued)*
defined 209
directory contents 11
editing 38
file permissions 15
file pool install 88
file space planning 72
finding dumps 128
introduction 7, 181
LOADBFS table 96
minidisks per server 81
multiple servers 98
non default setup 80
overview 75
path name syntax 39
path resolution 80
root file space in VMSYS 79
semantics 7
server CP directory sample 84
space considerations 94
tables 96
users per server 81
BFS LOADBFS 80
BFS LOADBFS, customization 89
BFS server
installing 75
user storage 83
bfs, CMS pipeline device driver 66
BFSDEL XEDIT 68, 195
bfsdirectory, CMS pipeline device driver 66
BFSEXAM XEDIT 68, 194
BFSLINE
LRECL option 42
XEDIT option 40
BFSLIST
BFS 181
EXEC 68
filelist for BFS 68
introduction 69
source code 192
XEDIT macro 68, 193
BFSPACE, monitor file spaces 112
bfsquery, CMS pipeline device driver 67
bfsreplace, CMS pipeline device driver 67
BFSROOT EXEC 88
BFS installation tool 79, 89
customization 80
shell install 94
bfsstate, CMS pipeline device driver 67
bfsxecute, CMS pipeline device driver 67
bibliography 203
bin
a BFS directory 10
default user 143
binary (FTP command) 55
bits, permission 38
blanks, trailing 42
BLOCKS
enrolling user 95
size of BFS disk 87
BPX CSL calls
BPX1GEG 150
BPX1GEU 150
BPX1GGI 150, 165
BPX1GGN 150, 165
BPX1GGR 150
BPX1GID 150
BPX1GPN 150, 165
BPX1GPU 150, 165
BPX1GUG 150
BPX1GUI 150
BPX1MNT 7
braces, code page 31
brackets, square, code page 31
broadcasting 154
BSD UNIX
general information 5, 44
printing 44
built-in arithmetic, shell programming 168
Byte File System
See BFS

C

C for VM/ESA compiler 6
C for VM/ESA service 136
C programming language, UNIX 5
C run-time library 8
C/370 8
C/VM, program product 133
callable services library
OpenEdition 133, 149
cancel command 44
canonical input 31, 209
carriage-return character, control-M 31
case sensitivity 21
case statement, shell script 176
cat (shell command) 31
catalog minidisk
file pool server 76
initial size 83
cd (FTP command) 55
cd (shell command) 53
CDPATH (environment variable) 28
CEEBLSPA, C library 133
CEEBLSPB, C library 133
CEL, C language library 133
cent sign 30
CENV group, GLOBALV environment variables 26
character special file, defined 209
character translation 31
characters
control 30
end-of-line 40
ERASE 31
ESCAPE 34

characters *(continued)*
KILL 31
LINEDEL 34
LINEND 34
new-line 40
portable 30
POSIX 41
QUIT 31
tab 42
chgrp (shell command) 53
chmod (shell command) 22, 53
chown (shell command) 53
CLIBNAMES statement 93
CLINKNAME statement 93
close (FTP command) 55
closing files 176
cluster format, CP directory 141
CMS 135
CMS short name 114, 120, 123, 127, 209
logical segment 129
physical segment 129
saved segment 129
saved segment, application code 129
segment 129
shared segment 129, 131
cms (FTP command) 55
CMS command, IDENTIFY 94
CMS pipelines, OpenEdition extensions 66
CMSEXEC, external link type 99
CMSFILES, file pool control statement 86
code page
1407 31
defined 209
editing 41
translation using pax 61
CODE, external link type 99
COLUMNS (environment variable) 28
columns, setting in the shell 34
combining commands, shell programming 178
command alias 109, 209
command interpreter, defined 209
command mode 36
command processing, shell 21
command substitution, in shell scripts 169
command, IDENTIFY 94
commands
.SHOW LOCK USER 115, 116
.SHOW LOCK WANTLOCK 115, 116
accept 44
BFSROOT 79
cancel 44
cat 31
cd 53
chgrp 53
chmod 22, 53
chown 53
cp 53
DEFSEG 129
commands *(continued)*
DELETE USER (OPENVM) 97
DFSMS ALTER 123, 125
DFSMS MANAGE 124, 125
DFSMS RECALL 127
DFSMS REPORT 126
DIRECTXA 148
disable 44
echo 45, 177
ed 24
enable 44
enroll user 95
env 23
exit (shell command) 18
FILEPOOL BACKUP 114, 116, 118
FILEPOOL FILELOAD 114, 119
FILEPOOL LIST BACKUP 114, 119
FILEPOOL MINIDISK 83, 95
FILEPOOL RELOAD 114
FILEPOOL RELOAD FILES 119, 120
FILEPOOL UNLOAD 114, 116, 117
FILESERV MINIDISK 83
find 128
functions 52
history 24
id 139
IDENTIFY 139
intuitive 36
Line 38
ln 52
lp 44, 45
lp (shell command) 45
lpadmin 44
lpc 44
lpd 44
lpmove 44
lpq 44
lpr 44
lprm 44
lpstat 44
lpusers 44
ls 21, 52, 53
ls -ils 127
ls -l 21
ls -r 21
ls -ri 52
ls (shell command) 19
mailx 166
mkdir 52
mv 53
OPENVM 38, 46
OPENVM CREATE DIRECTORY 52
OPENVM CREATE EXTLINK 50, 52
OPENVM CREATE LINK 52
OPENVM CREATE SYMLINK 52
OPENVM ERASE 50, 53
OPENVM EXTLINK 99
OPENVM GETBFS 48, 50, 53, 129

commands *(continued)*
OPENVM LIST 52, 123
OPENVM LIST example 90
OPENVM LISTFILE 53
OPENVM MOUNT 53, 102
OPENVM OWNER 53
OPENVM PARCHIVE 53
OPENVM PERMIT 53
OPENVM PUTBFS 50, 53
OPENVM QUERY DIRECTORY 53
OPENVM QUERY LINK 53
OPENVM QUERY MASK 53
OPENVM QUERY MOUNT 53
OPENVM RENAME 53
OPENVM RUN 46, 53, 131
OPENVM SET DIRECTORY 53
OPENVM SET MASK 53
OPENVM shell 17, 18, 131
OPENVM UNMOUNT 53
pax 53
print 44, 45, 177
printf 177
purge printer 45
PUTBFS 50
pwd 53
QUERY FILEPOOL CONFLICT 115
query printer 45
r 24, 25
reject 44
rm 53
rmdir 53
Scrolling 38
sed 24
SEGMENT LOAD 131
SET FILEWAIT ON 115
SET INPUT 33
SET OUTPUT 31
shell, return 177
shell, set 170
shell, test 174
shell, typeset 172
stty 31
su 137
td (shell command) 177
TERMINAL 34
TYPE 31
umask 22, 53
xedit shell function 178
common execution library (CEL) 133
communication services 132
compiler 6
component names 135
COMPRESS, XEDIT subcommand 42
conditional expressions, shell 174
configuration, SHELL LOADBFS 96
considerations
line mode 34
mailx 166
console statement, file pool install 85
contents, of BFS directory 10
CONTROL BACKUP, file pool installation 87
control characters 30
control minidisk
determining size 81
file pool server 76
sizes 82
table 82
control program 44, 45
See also CP (control program)
CONTROL RELOAD file 120
control structures 174
control-H, backspace character 31
control-M, carriage-return character 31
coordinated resource recovery (CRR) 78
Coordinated Universal Time 110
COR, corrective service 134
Corrective Service 134
count BFS blocks
DU 69, 182
CP (control program)
directory, reporting tool 183
directory, settings for BFS server 85
directory, statements 140, 141
discussion 135
cp (shell command) 53
CP directory reporting tools 151
CPI communications, library 134
cpio (shell command) 58
CREATE, OPENVM EXTLINK 99
creating files 34
CRR 78
CS, communication services for VMplex 132
CSL 149
OpenEdition 133, 149
OpenEdition libraries 133
CSLPSCAN
directory reporting tool 151, 183
current directory, defined 209
current mount point 101
current working directory, defined 209
cursor movement keys 36
customized install 80

D

daemon, default user 143
DASD, BFS storage 81
data
migration 50
migration, pax 61
sharing, OpenEdition to OpenEdition 132
Data Facility Storage Management Subsystem for VM 122
data spaces, file pool requirements 85
database
RACF 164
user 146

DCE 2
  DCEBASE, component 135
  DCEPRIV, component 135
  Directory Services Client 1
  interoperability 2
  Remote Procedure Call 1
  RPC 1
  Security Services Client 1
  Threads 1
debug (FTP command) 55
debugging aid 179
debugging, shell programs 179
default
  BFS servers 78
  directory, defined 209
  file pool and servers 79
  permissions, file system 11
  users 143
  value 146
defining a BFS server 83
defining POSIX user groups 73
DEFSEG, command 129
delete (FTP command) 55
DELETE USER, OPENVM command 97
deleting user 157
delimit (FTP command) 55
desktop publishing programs 34
determine control minidisk 81
dev, a BFS directory 10
DFSMS/VM
  CMS short name 123, 127
  commands
    ALTER 123, 125
    MANAGE 124, 125
    RECALL 127
    REPORT 126
  considerations 123
  fully qualified name 123
  inodes 123, 127
  Interactive Storage Management Facility 123
  introduction 3, 118, 122
  ISMF 123
  PTF required for DIRMAINT 1.5 123
  regarding the BFS 122
DGTVAUTH DATA (Dirmaint) 123
dir (FTP command) 55
directory
  /tmp 35
  BFS contents 10
  BFS overview 77
  BFS server 84
  CP 10
  created by shell install 92
  defined 209
  initialization 141
  names 11
  parent 38
  permission bits 11
directory *(continued)*
  reporting tool, CSLPSCAN 183
  sample statements 141
  settings for BFS server 85
directory reporting tools 151
directory services client 1
DIRECTXA, CP command 148
DIRMAINT
  introduction 123
  PTF for DFSMS 123
DIRPOSIX
  for installation of BFS 79
  LOGFILE 144
  options 142
  UIDEXCL 142
  USEREXCL 142
  utility program 141
DIRPSCAN, directory reporting tool 152
disable command 44
disadvantages, dedicated BFS server 90
display device, for shell 29
Distributed Computing Environment 1, 2
distributed environment, file access 132
DMSOVB EXEC 188
DMSPARMS 94
  BFS example 86
  file sample 96
  server installation 83
document, authors xiv
download a file 45
DU sample program
  EXEC 70
  EXEC, source 198, 199
  introduction 69
  output 70
  shell script 70
  tool 69
dumb terminal 29
dumps
  defined 209
  finding 128
  locating 128
duplicate, commands, POSIX shell 94
duplicated UIDs 143
DV 135

E

EBCDIC
  code page translation 31
  defined 209
ebcdic (FTP command) 55
echo (shell command) 45, 170, 177
echo $? 170
ed
  as an editor 35
  use in history 24
editing files, using XEDIT shell script 38

editing, using xedit for BFS 178
EDITOR (environment variable) 28
editors
  ed 35
  EDLIN 35
  emacs 34, 36
  for the shell 34
  full-screen 36
  line 35
  macros 36
  sed 36
  text 34
  vi 36
  XEDIT 38
effective
  GID 139
  UID 137, 139, 146
  UID defined 209
emacs
  editor 34, 36
enable command 44
end-of-line character 40
enroll
  ENROLL USER, command 95
  in BFS 94
  ROOT user ID 89
  TMP user ID 89
ENROLL VAR 89
env (shell command) 23
environment variables
  - character 28
  CDPATH 28
  COLUMNS 28
  EDITOR 28
  ENV 28
  FCEDIT 28
  GLOBALV 26
  HISTFILE 28
  HISTSIZE 28
  HOME 28
  IFS 29
  LANG 29
  LC_ALL 29
  LE/370 133
  LINENO 29
  LINES 29
  LOGNAME 29
  LPDEST 45
  MAIL 29
  MAILCHECK 29
  MAILPATH 29
  MBOX 29
  OLDPWD 29
  PATH 29
  PID 29
  PS1-PS4 29
  PWD 29
  RANDOM 29
environment variables *(continued)*
  SECONDS 29
  SHELL 29
  tilde 28
  TMOUT 29
  TZ 27, 29
er, alias 109
ERASE char 31
err, signal 179
errors during BFS server generation 96
errors, during installation 96
ESCAPE character 31, 34
escape sequences
  shell input 30
  terminals 29
etc, a BFS directory 10
examples
  BFS server CP directory 84
  OPENVM commands 46
  OPENVM MOUNT command 102
  shell 138
  shell script 180
  x shell script for editing 38
EXEC_SETIDS, planning 73
exec(), security aspects 165
EXECs
  BFSLIST 68
  BFSLIST source 192
  BFSROOT 88
  DU 69
executable commands 10
execute, file permission 13
execution paths 22
exit 178
EXIT trap 180
exiting the shell 18
EXPAND, XEDIT subcommand 42
export variables 168
export, in shell script 168
expression 176
extended security controls 164
extensions, REXX 180
external links
  between default file pool servers 80
  creating 99
  defined 209
  home directories 101
  overview 99
  use of 7
  used in installation 80
External Security Manager 164
EXTLINK
  /tmp 89
  /var 89
  BFS default use 89

F

failure of program 171
FCEDIT (environment variable) 28
feature, Shell and Utilities 91
fi, shell end if 175
FIFO special file, defined 210
file access, distributed environment 132
file archive
  cpio 58
  pax 59
  tar 57
file pool
  administration machine 77
  BFS use 75
  data spaces 85
  defined 210
  generation 86
  installation 78
  overview 75, 76
  planning 78
  repository 83
  selecting name 87
  structure 76, 77
file pool servers
  default for BFS 78
  directory settings 85
  installation settings 81
  minidisks used 76
  multiple install 98
  overview 78
  profile 88
file search 180
file server, estimating users 81
file space
  defined 210
  monitoring consumption 112
  planning 72
file system
  backups 113
  concurrent 116
  considerations 114
  creating backups 50
  implementation 7
  locks 114
  methods 114
  permissions 11
  planning 78
  restoring individual files 119
  root 146
file time stamps 11, 20
file transfer
  FTP 54
FILELIST, for BFS 68
FILEPOOL commands
  BACKUP command 114, 116, 118
  FILELOAD command 114, 119
  LIST BACKUP command 114, 119
  MINIDISK command 83, 95
FILEPOOL commands *(continued)*
  RELOAD command 114, 119, 120
  UNLOAD command 114, 116, 117
files
  closing 176
  descriptors 177
  down-load 45
  ID, mixed case 39
  mixed-case 43
  mode, defined 210
  names 11
  open 176
  permissions
    defaults 17
    root 137
    security of 162
    table of 15, 22
  reading from 176
  type, defined 210
FILESERV
  command, generate file pool 78
  file pool generation 86, 87
  MINIDISK, BFS server command 83
  POOLDEF, example 87
  START, file pool start 88
find (shell command) 128
finding dumps 128
first time handling of directory 141
fl, alias 109
for
  shell script programming 175
  shell script statement 175
foreground process, defined 210
FS, shell script sample program 180
FSROOT 72
FSROOT, how to define 147
FTP
  commands 54
  file transfer 54
full-screen
  CMS, and shell 43
  editor 36
  layout 36
fully-qualified path name 38, 123
function keys 36, 210
function to edit files 109
function, setlocale() 8
functions, shell 177

G

GCS 135
general preparation, for OpenEdition 72
generate, file pool 86
get (FTP command) 55
getgrgid() 73
getgrgid(), function call 165
getgrnam() 73

getgrnam(), function call 165
getpwnam(), function call 165
getpwuid(), function call 165
GID 147
  defined 210
  directory 143
  effective 139
  explained 139
  planning 73
  strategy 139
global definition section, CP directory 145
global system settings, SYSTEM CONFIG 166
GLOBALDEFS
  directory statements 141, 145
GLOBALOPTS, CP directory 145
GLOBALV
  CENV group 26
  environment variables 26
  SELECT, setting variables 28
glossary 209
GNAME 147
GNAME, in directory 143
Gretchen trick 109
group
  enrolling user 95
  primary 147
group ID
  defined 210
group name 146
group name, defined 210
GUI 135

H

hello world (shell script) 22
help
  facility 36
  FTP command 55
  getting it in the shell 26
  online 112
  OPENVM commands 112
  OSHELL 26
  shell commands 112
HEXOUT, sample program 32
high-level languages 149
highlighting 112
hints and tips, DCE, BFS initialization 94
HISTFILE
  shell variable 24, 28, 111
history (shell command) 24
history file 24, 111, 210
history, of UNIX 5
HISTSIZE (environment variable) 28
Holding, screen state 30
HOME (environment variable) 28
home directory
  creating 153
  defined 210
  using with links 101
hung session 34
HX immediate command 34

I

i, shell script variable 175
I/O Mechanism, shell programming 168
IBM Language Environment for MVS and VM C-Runtime 133
ICANON 31
ICKDSF 135
id (shell command) 139
IDENTIFY (CMS command) 94, 139
IEEE POSIX 6, 7
if, conditional shell expression 174
IFS (environment variable) 29, 169
immediate command, HX 34
implementation, file system 7
in, shell script 175
inconsistency of UNIX command options 21
initial
  BFS server DASD allocation 81
  user program 146
  working directory 146
INITLZ, directory initialization 143
inodes 114, 120, 123, 127
  defined 209, 210
input mode 36
input, canonical 31
installation
  BFS 75
  BFS LOADBFS table 80
  BFSROOT 79
  creating home directories 153
  customized 80
  customizing shell startup 93
  file pool 78
  OpenEdition 75
  restoring 96
  Shell and Utilities Feature 91
  tailoring system configuration 166
Institute of Electrical and Electronics Engineers, POSIX 6
intelligent workstation 29
Inter-System Facility for Communication, remote BFS access 132
Interactive Storage Management Facility 123
Internal File Separator 169
interoperability 1, 2
introduction
  byte file system 7
  UNIX terms 5
intuitive commands 36
ISFC, remove BFS access 132
ISMF 123
IUPGM, how to define 147
IWDIR, how to define 147

J

jobs, printing 44
Julian day 111

K

kernel
  defined 210
  regarding security 161
keyboard sequences 30
keys
  cursor movement 36
  function 36
kill character 31
kill, defined 210
Korn Shell, overview 17

L

LANG (environment variable) 29
languages, national 41
large-volume printing 45
LC_ALL (environment variable) 29
LE/370
  environment 133
  introduction 135
  saved segments 133
  service 136
lib, a BFS directory 10
line
  commands 38
  editor 35
  mode considerations 34
  mode, defined 210
LINEDEL character 34
LINEND character 34
LINENO (environment variable) 29
LINES (environment variable) 29
LINK, setting up BFS server 85
links, external 99
ln (shell command) 52
LOAD subcommand in XEDIT profile 39
LOADBFS
  BFS 89
  customization 89
  installing the shell 92
  SHELL 92
  shell tables 96
local service 134
locale, defined 210
locating dumps 128
locks, file system 114
locstat (FTP command) 55
log files 154
log minidisk
  file pool server 76
  regarding file pool server 82
LOGFILE, DIRPOSIX 144
logging 154
logical segment 129
login
  as ROOT 137
  authentication 162
  defined 210
  to shell 131
LOGNAME (environment variable) 29
loops
  in shell 175
  while 175
lowercase, string conversion 168
lp (shell command) 44, 45
lpadmin (shell command) 44
lpc (shell command) 44
lpd (shell command) 44
LPDEST (environment variable) 45
lpmove (shell command) 44
lpq (shell command) 44
lpr (shell command) 44
lprm (shell command) 44
lpsched (shell command) 44
lpstat (shell command) 44
lpusers (shell command) 44
ls 74
  -ils option 127
  -l option 21
  -r option 21
  FTP command 55
  shell command 19, 21, 52, 53

M

machine, administration 85
MAIL (environment variable) 29
MAILCHECK (environment variable) 29
MAILPATH (environment variable) 29
mailx 74
  considerations 166
  security 164
  shell command 166
MAINT 493 minidisk 141
management classes 123
MAXDISK, file pool setting 81
MAXDISKS, file pool control statement 87
MAXUID, DIRPOSIX option 142
MAXUSERS
  file pool control statement 87
  file pool setting 81
MBOX (environment variable) 29
mdelete (FTP command) 56
MEL, external link type 99
messaging 154
mget (FTP command) 56
migrating data, pax 61
minidisk
  control 81
  control (file pool server) 76

minidisk *(continued)*
file pool control statement 85
FILEPOOL 95
log (file pool server) 76
maximum number of per server 81
MINIOPT, file pool server statement 85
MINUID, DIRPOSIX option 142
mixed-case files 21, 39, 43
mkdir (shell command) 52
mode
bits 15
command 36
defined 210
FTP command 56
input 36
modern printers 45
MODULE list, POSIX shell 94
monolithic format, CP directory 141
More, screen state 30
Mortice Kern Systems Inc, shell 1
mount external link 99, 104
mount point
BFS 100
defined 210
mount, defined 210
MOUNT, external link type 99
mput (FTP command) 56
Multics 5
multiple BFS servers, installing 75
multiple file search 180
multiple user mode, file pool server 88
multitasking library 134
mv (shell command) 53

N

NAME_MAX 8
NAMETYPE BFS, XEDIT option 39
NAMETYPE CMS, XEDIT option 39
national languages, editing 41
new-line character 40
NEWLINE 169
nickname (CMS) 45
NOBACKUP, file pool control statement 86
nobody, default user 143
node.user 45
NODFSMS, file pool control statement 87
noexec, shell trace option 179
NOMDC, file pool server control statement 85
NOMDCFS, file pool server option 84
noncanonical mode, defined 211
nonprogrammable terminal 29
noop (FTP command) 56
normal exit 171
Not Accepted, screen state 30

O

octal, file permissions 12, 13
offloading BFS data 50
offset 111
OLDPWD (environment variable) 29
online help 112
OPENVM commands 112
shell commands 112
open (FTP command) 56
Open Software Foundation, adoption of POSIX 6
OpenEdition
BFS overview 75
callable services library routines 133
CSL interface 133
installing BFS 75
introduction 1
planning 71
POSIX implementation 132
primer 5
printing 44, 45
privileged users 137
programming 66
reviewing security 162
security 159
servicing 134
Shell and Utilities Feature 91
terminology 7
UID and GID setup 145
user database reporting tool 151
opening files 176
OPENVM commands
CREATE DIRECTORY, command 52
CREATE EXTLINK, command 50, 52, 99
CREATE LINK, command 52
CREATE SYMLINK, command 52
ERASE, command 53
functions 52
GETBFS, command 48, 53, 129
help 112
in editing session 38
limitations 52
LIST, command 52, 53, 123
LIST, showing default directories 90
LISTFILE 73
MOUNT, command 7, 53, 102
OWNER, command 53
PARCHIVE, command 53
PERMIT, command 53
PUTBFS, command 53
QUERY DIRECTORY, command 53
QUERY LINK, command 53
QUERY MASK, command 53
QUERY MOUNT, command 53, 101
RENAME, command 53
RUN 99
RUN, command 46, 53, 131
SET DIRECTORY, command 53
SET MASK, command 53

OPENVM commands *(continued)*
SHELL, command 17, 131
summary 46
UNMOUNT, command 53
OPENVM GETBFS
binary data 50
modules 50
text data 50
opt, a BFS directory 10
options
BFSLINE 40
CP directory 140
MAXCONN, setting BFS connections 84
OS, simulation under CMS 132
OSF, regarding POSIX 6
OSHELL, help menu 26
output file, defined 211
output redirection, defined 211
overview, file pool 76
OVM segment 164
owner, defined 211

P

paging, amount of 35
parameter modifier 170
parameters
BFS startup 87
shell 170
parent directory 38
parent directory, defined 211
pass (FTP command) 56
PATH (environment variable) 29
path name
absolute 38
current 10
defined 211
fully-qualified 38
names 38
relative 38
resolution, running a script 22
search order 22, 23
syntax, BFS 39
updating path variable 23
PATH_MAX 8
pax
creating a backup 50
example of backup 50
pax (shell command) 50, 53, 59
permissions
default 17
file system 11
permission bits 11, 38
permission bits, advanced discussion 15
permission mode, defined 211
read for editing 38
search for editing 38
write for editing 38
physical screen 36
physical segment 129
PID (environment variable) 29
pipe, defined 211
pipelines, OpenEdition extensions 66
pipes in the shell 178
planning
file pool 78
file servers 78
file system 78
for OpenEdition 71
installation 134
OpenEdition installation 72
policies, security 159
POOLDEF
file 96
server installation 83
statement, file pool server 96
portability 1
portable character set 11, 30
portable character set, defined 211
Portable Operating System Interface for Computing 1
positional parameter, shell scripts 169
POSIX
.profile 107
alias 209
application code, saved segment 129
application portability 1
applications 131
applications, saved segment 129
characters 41
defined 9, 211
group ID 146
HISTFILE 24, 111
IEEE 7
implementation 132
Institute of Electrical and Electronics Engineers 6
login shell 131
OpenEdition compliance 1
POSIX shell
alias 109
command processing 21
history file 24, 111
profile variables 109
prompt, highlighting 112
prompts 112
saved segment 129
variable, TMOUT 109
profiles 107
programming interfaces 133
shell variables
standards
1003.1 1
1003.1a 1
1003.1c 1
1003.1d 1
1003.2 Shell and Utilities 1, 17

POSIX *(continued)*
terminology 7
time zone 110
TZ 110
user database 145
user ID 146
user ID, planning 72
user profile 109
values in CP directory 140
POSIX exec(), planning 73
POSIX option reporting tools 151
POSIX.1 support 75
POSIX(ON) pragma 133
POSIXGLIST, CP directory statement 148
POSIXGROUP
CP directory statement 73, 145
POSIXINFO
CP directory statement 72, 139, 146
POSIXOPT
CP directory 149
QUERYDB 164
SETIDS ALLOW 85
POSXSOCK, library 134
potential conflicts, DIRPOSIX 144
prefixing
paths 22, 23
primary group 139, 147
PRIMarygroup, DIRPOSIX option 142
print (shell command) 177
printing
BSD 44
facilities 44
large-volume 45
OpenEdition 45
print command 44, 45
printers 44, 45
printf (shell command) 177
printing jobs 44
queue 44
system V 44
UNIX 44
process, defined 211
Product Service Update (PSU) 134
profile
file pool server 88
POSIX shell 107
profile user 109
profile variables 109
PROFILE XEDIT 39
program directory 134, 141
program source code 183
Program Temporary Fix 134
program-to-program communications functions 134
programs
BFSROOT 79
DU source 198
OpenEdition 66
shell script 167
programs *(continued)*
shell script samples 180
prompt
defined 211
highlighting 112
shell 112, 139
shell, $ 138
PS1-PS4 (environment variable) 29, 112
PTF 134
PTF required for DIRMAINT and DFSMS 123
purge printer (CP command) 45
put (FTP command) 56
PWD (environment variable) 29
pwd (FTP command) 56
pwd (shell command) 53

Q

QUERY FILEPOOL CONFLICT, (CMS command) 115
query printer (CP command) 45
QUERYDB, planning 73
queue
printer 44
spooling 44
QUICKDSP, file pool option 84
quit (FTP command) 56
QUIT char 31
quote (FTP command) 56
quoted string operands 146

R

r (shell command) 24, 25
RACF 140, 164
access checking 164
audits file 164
database 164
defined 211
RANDOM (environment variable) 29
re. alias 109
read
file permission 13, 38
reading from files 176
reading from terminals 176
recovery, after installation 96
regular expressions, shell scripts 171
regular file
defined 210
editing 38
reject command 44
relative path name 38
relative path name, defined 211
remote BFS access 132
remote procedure call 1
rename (FTP command) 56
REPlace, DIRPOSIX option 142
reporting
CP directory tool 151, 183

Resource Access Control Facility, defined 211
restoring
  BFS after building the POSIX shell 96
  BFS after execution of BFSROOT 96
  environment after installation errors 96
restoring individual files 119
retrieve, a command in the shell 25
return (shell command) 177
return value, shell script 171
reviewing system security 162
reviewing the directory 151
REXX
  REXX extensions for OpenEdition 66, 135, 180
rm (shell command) 53
rmdir (shell command) 53
root 137
  default user 143
  defined 211
  definition of 137
  directory, defined 211
  file space in BFS 79
  VMSYSU BFS extensions 80
  what is it? 137
ROOT, ENROLL 89
RPC, OpenEdition feature 1
RPIDIRCT EXEC 164
RSCS 45
Running, screen state 30

S

S/390 and UNIX 1
sample program
  DU 69
  DU source 198
  HEXOUT 32
  output from UNLOADBF EXEC 97
  shell scripts 180
  source code 68, 183
saved segment
  application code 129
  LE/370 133
saved set-UID 146
SAVESEGID, file pool control statement 86
scalability, VM/ESA 2
SCEE, C library 133
SCEELKED TXTLIB 133
SCEERUN LOADLIB 133
SCEEX, C library 133
SCIF, file pool monitoring 85
screen state
  Holding 30
  More 30
  Not Accepted 30
  Running 30
  terminals 30
  VM READ 30
script programming 167
script, samples shell 180
scrolling commands 38
search permission for editing 38
second argument string, in XEDIT profile 39
SECONDS (environment variable) 29
security
  basic concepts 159
  extended controls 164
  file permission bits 12
  how much 160
  kernel mode 161
  policies 159
  programs that operate with root authority 161
  reviewing system 162
  supervisor state 161
  UNIX 159
  user mode 161
  who needs 160
Security Services Client 1
sed (shell command) 24, 36
segment 129
SEGMENT LOAD, command 131
selecting additional security features 73
semicolon, shell programming 178
sendport (FTP command) 56
sendsite (FTP command) 56
server PROFILE, example 88
servername DMSPARMS 86
servers
  BFS 78
  BFS definitions 83
  default file pool 78
  installing BFS 75
  multiple 98
service
  OpenEdition 134
  user IDs 136
servicing
  local 134
  recommendations 135
  types 134
session, defined 211
set (shell command) 170
SET commands
  FILEWAIT ON 115
  IMAGE, XEDIT subcommand 42
  INPUT 33
  OUTPUT 31
set-group-ID mode bit, defined 211
set-ID, functions 139
set-user-ID 166
set-user-ID mode bit, defined 211
setlocale() function 8
SFS 75, 122
  DMSPARMS file 96
  POOLDEF file 96
  regarding BFS 7, 122
  regarding BFS installation 75

SFS *(continued)*
regarding CSL library 134
SFSDOT LIST3270 115
SFSDOT LIST3720 115
sgid
permission bit 15, 16
SHARE REL 85
Shared File System
See SFS
shared segment
using 129, 131
shell 22
alias 109, 209
attributes 173
basic constructs 167
case sensitivity 21
command option examples 21
command processing 21
control structures 174
customizing startup 93
defined 211
definition of 18
directories created 92
display device 29
ed editor 35
editors 34
emacs editor 34
escape sequences 30
example of 138
exiting 18
full-screen CMS 43
functions 177
history file 24, 111
how to free up a hung session 34
input case 21
install, BFSROOT EXEC 94
installation 91
introduction 17
line mode considerations 34
pipes 176
planning 72
planning for install 72
POSIX 1003.2 17
prefixing 23
printing 45
profiles 107, 109
prompt 138, 139
prompt, highlighting 112
prompts 112
running a script 22
sample 181
samples script 180
setting columns 34
setting time zone 27
starting 18
terminal 29
tricks 22
variables 109
shell *(continued)*
vi editor 36
XEDIT editor 36
SHELL (environment variable) 29
shell and utilities
See shell
shell commands
cat 31
cd 53
chgrp 53
chmod 22, 53
chown 53
cp 53
cpio 58
echo 170
ed 24
env 23
find 128
help 112
history 24
ln 52
ls 19, 21, 52, 53
ls -ils 127
ls -l 21
ls -r 21
ls -ri 52
mailx 166
mkdir 52
mv 53
pax 50, 53, 59
pwd 53
r 24, 25
rm 53
rmdir 53
sed 24
set 170
starting shell 18
su 137
tar 57
typeset 172
umask 22, 53
shell function, xedit for editing 178
SHELL LOADBFS 96
SHELL LOADBFS (install file) 92
shell script 167
defined 211
looping 175
programming 22, 167
samples 180
variables 168
x 38
x for XEDITing files 24, 38
xedit function 109
shell variable
HISTFILE 24, 111
path 23
timezone 110
TMOUT 109

shell variable *(continued)*
TZ 110
shell variables, value 169
SHELL/U 135
signals 179
err 179
trap 179
single BFS server 75, 80
Single Console Image Facility 85
See also SCIF, file pool monitoring
site (FTP command) 56
Size of File Pool 82
SMS 122
socket, defined 211
source code, program 183
special files, defined 210
special parameter 170
spooling daemon 44
spooling queues 44
square brackets, code page 31
staff, default user 143
standard error 177
standard input 176
standard output 176
start-up, BFS server 83
starting the shell 18
startup parameter, BFS 87
statements, CP directory 140
status (FTP command) 56
stderr 177
stdin 177
stdout 177
sticky bit 15, 16
sticky bit, defined 211
storage group, defined 211
storage groups
BFS server 83
regarding BFS installation 75
storage management 122
storage, user groups overview 76
string lengths, in shell scripts 169
string search facilities 36
struct (FTP command) 56
structures
combination of 175
control 174
stty (shell command) 31
stty operands 31
su (shell command) 137
su UIDs 138
subcommand
LOAD in XEDIT profile 39
SET CASE UPPERCASE 43
XEDIT 38
substring operators, shell programming 168
suid
permission bit 15, 16
sunique (FTP command) 56
superuser
defined 211
definition of 137, 138
prompt example 138
state, regarding security 161
support, POSIX.1 75
SVMSTAT, file pool option 85
symbolic link, defined 212
symbolic link. 10
sys, default user 143
SYSAFFIN
CP directory 145
CP directory statement 144
SYSentries, DIRPOSIX option 142
SYSPROF EXEC, regarding .profile 107
system
administration 107
BFS, .profile 107
default user 143
dumps 128
FTP command 57
performance 129
SYSTEM CONFIG
global POSIX settings 34, 166
system ID, on terminal 30
system managed storage 122
SYSTEM SEGID 129
System V
introduction 5, 17, 44
printing 44

T

tab
characters 42
pertaining to shell 169
using XEDIT 42
tar (shell command) 57
TCP/IP, FTP 54
td (shell command) 177
temporary user mount point 103
terminal 29
command 34
line mode considerations 34
nonprogrammable 29
reading from 176
UNIX 29
writing to 177
terminology, POSIX 7
test (shell command) 174
testfile 182
text editors 34, 35
threads 1
tilde, (environment variable) 28
time stamps, files 20
time zone 110
offset 111
setting in shell 27, 110

TMOUT (environment variable) 29, 109
TMP
  BFS file space 80
  ENROLL 89
tmp, a BFS directory 10
tools
  BFSDEL XEDIT 68, 195
  BFSEXAM XEDIT 68, 194
  BFSLIST 68, 69
  BFSLIST EXEC 68
  BFSLIST EXEC source 192
  BFSLIST XEDIT 68, 193
  EXECs, BFSLIST 68
  FILELIST 68
  for OpenEdition 68
tracing shell scripts 134
trailing blanks 42
transparent services access facility, remote BFS access 132
trap
  EXIT 180
  signals 179
tree search, sample shell program 181
tricks for the shell 22
TS, sample shell program 181
TSAF
  remote BFS access 132, 135
type (FTP command) 57
TYPE command 31
typeset
  shell command 172
  type attribute 172
TZ
  (environment variable) 27, 29, 110

U

u, a BFS directory 10
UID 72, 139, 146
  defined 212
  duplicated 143
  effective 137, 139
  for installation of BFS 79
  of zero 146
  planning 72
  real 139
  strategy 139
umask (shell command) 22, 53
unique file pool, generation 88
UNIX
  Bell Labs′ contribution to 5
  BSD V4 5
  case sensitivity 21
  command option consistency 21
  directory names 11
  file names 11
  file permissions 22, 137
  history 5
  IEEE POSIX standard 6
UNIX *(continued)*
  Institute of Electrical and Electronics Engineers 6
  introduction 20
  Korn Shell 17
  login, as ROOT 137
  mixed case 21
  Open Software Foundation 6
  OSF POSIX standard 6
  path search order 22
  POSIX 1003.2 standard 17
  primer 5
  printing 44
  root 137
  security 159
  shell 17
  shell scripts 22
  shell variable, path 23
  superuser 137, 138
  System V 5
  terminal 29
  text editors 34, 35
  tricks 22
UNLOADBF EXEC
  overview 97, 196
  sample output 97
untokenized 39
uppercase
  during editing 43
  string conversion 168
user
  administration 137
  database 146
  enrolled in BFS 94
  estimating for file server 81
  FTP command 57
  mount points 100
  profile 109
  profile variables 109
  profile, .profile 107
  shell variable, HISTFILE 24, 111
  shell variable, time zone 110
  shell variable, TZ 110
  shell, alias 109
  shell, command processing 21
  shell, history file 24, 111
  shell, prompt, highlighting 112
  shell, prompts 112
  storage groups, BFS overview 76
user database reporting tools 151
user direct (CP)
  CP directory 141
  setup 145
user groups
  for installation of BFS 79
  planning 73
user ID
  defined 212

user IDs, service 136
user minidisk, BFS server 83
user POSIX option reporting tools 151
user-friendly 36
user's real UID 146
USER_DEFAULTS, directory statement 164
user, deleting 157
USERS, setting BFS connections 84
usr, a BFS directory 10
UTC 110

V

VAR
  BFS file spaces 80
  ENROLL 89
  in shell script 168
var, a BFS directory 10
variables
  expansion 170
  export 168
  record format 42
  shell scripts 168
  typeset 172
verbose trace 179
vi editor 36
VM collection, data sharing 132
VM READ, screen state 30
VM SYSBLDS 134
VM/ESA
  POSIX implementation 132
  scalability 2
  the system of choice 2
  Version 2, OpenEdition introduction 1
VM60499, APAR for Dirmaint 1.5 and DFSMS 123
VMBFS
  dedicated BFS server 85
  file pool server 89
VMFINS, overview 134
VMLIB, OpenEdition library 134
VMLINK NAME file, shell startup 93
VMMTLIB 149
  CSL OpenEdition library 133
  VM/ESA library 134
VMSERBFS, dedicated BFS server 84
VMSERVR, relation to BFS 78
VMSERVS DMSPARMS 123
VMSERVS, relation to BFS 78
VMSERVU
  relation to BFS 78, 80
VMSES/E 134, 135
VMSYS 79, 89
  BFS root file space 79
  relation to BFS 78
VMSYSR, relation to BFS 78
VMSYSU 80, 89
  BFS root extensions 80
  relation to BFS 78

W

while Loop 175
white space 129
white space, defined 212
word processors 34
work disk, file pool server 83
working directory, defined 212
workstation
  affinity 3
  intelligent 29
wrapping of lines 36
write
  file permission 13, 38
writing to terminals 177

X

x shell script 38
XCONFIG ADDRESPACE, regarding BFS server 85
xedit
  CMS command, use in history 24
  editing a BFS file 38
  features 36
  function for the shell 109, 178
  options to edit BFS files 40
  shell editor 36
  subcommands 38
  tabs using 42
  to manipulate stored commands 24
  XEDIT macros
    BFSDEL 68, 195
    BFSEXAM 68, 194
    BFSLIST 68, 193
XEDIT, sample shell script 38
xtrace option verbose 179
xtrace, shell programming 179

Z

zero, UID 146

Printed in U.S.A.