Prime®

PRIMOS® User's Guide

Revision 22.0

DOC4130-5LA

PRIMOS® User's Guide

Fifth Edition

Stephen Lewontin

This guide documents the software operation of the Prime Computer and its supporting systems and utilities as implemented at Master Disk Revision 22.0 (Rev. 22.0).

Prime Computer, Inc., Prime Park, Natick, MA 01760

Copyright Information

Printing History

First Edition (IDR4130) January 1980
Second Edition (PDR4130) December 1980
Third Edition (DOC4130-190) August 1982
Fourth Edition (DOC4130-4LA) April 1985
Update 1 (UPD4130-41A) January 1986
Update 2 (UPD4130-42A) July 1987
Fifth Edition (DOC4130-5LA) October 1988

Credits

Project Editor: Barbara Fowlkes

Project Support: Frank Calvillo, Joan Karp, Don Markuson, Betsy Perry

Design: Leo Maldonado

Graphics Support: Mingling Chang, Jeannette Monaco

Illustration: Mike Moyle

Document Preparation: Mary Mixon, Katherine Normington

Composition: Julie Cyphers, Sharon Temple

Production: Judith Gordon

How to Order Technical Documents

Follow the instructions below to obtain a catalog, a price list, and information on placing orders.

United States Only: Call Prime Telemarketing, toll free, at 800-343-2533, Monday through Friday, 8:30 a.m. to 5:00 p.m. (EST).

International: Contact your local Prime subsidiary or distributor.

Customer Support Center

Prime provides the following toll-free numbers for customers in the United States needing service:

1-800-322-2838 (Massachusetts)
1-800-541-8888 (Alaska and Hawaii)
1-800-343-2320 (within other states)

For other locations, contact your Prime representative.

Surveys and Correspondence

Please comment on this manual using the Reader Response Form provided in the back of this book. Address any additional comments on this or other Prime documents to:

Technical Publications Department
Prime Computer, Inc.
500 Old Connecticut Path
Framingham, MA 01701

Contents

	About This Book	xi
	Part I: PRIMOS Basics	
1	**Getting Started**	1-1
	The PRIMOS Operating System	1-1
	The Terminal	1-4
	The User's Dialog With PRIMOS	1-5
	The Command Line	1-7
	Essential PRIMOS Commands	1-10
2	**The PRIMOS File System**	2-1
	File System Objects	2-1
	Naming File System Objects	2-3
	The File System Tree Structure	2-5
	Pathnames	2-6
	Shortening Pathnames	2-8
	Pathnames, Objectnames, and PRIMOS Commands	2-10
3	**PRIMOS File System Commands**	3-1
	Examining Directory Contents	3-1
	Attaching to Another Directory	3-3
	Returning to Your Origin Directory	3-5
	Creating New Directories	3-5
	Examining the Contents of a File	3-6
	Renaming File System Objects	3-7
	Copying File System Objects	3-8
	Deleting File System Objects	3-11
	Protecting Objects From Accidental Deletion	3-12
	Recording Terminal Sessions	3-14
	Interrupting Commands	3-15

4 **Creating and Printing Files** 4-1
Text Files 4-1
Text Editors 4-2
Using EDITOR 4-5
Basic ED Commands 4-10
Printing Text Files With SPOOL 4-24

5 **Protecting Your Files and Directories** 5-1
An Overview of File Protection 5-1
Access Control Lists (ACLs) 5-3
Specific ACLs 5-5
Access Categories 5-9
Default Protection 5-14
Matching Access Rights 5-20
Who Can Set ACLs 5-20
Types of Access Rights 5-22
Priority ACLs 5-28

6 **Command-line Features** 6-1
Multiple Commands 6-1
Iteration 6-2
Wildcards 6-2
Treewalking 6-8
Name Generation 6-10
Syntax Suppression 6-13

7 **Command-line Editor** 7-1
Getting Started 7-1
Advanced ECL Commands 7-7
ECL Command Summary 7-24
ECL Command Options 7-28

8 **Customizing Your Environment** 8-1
Changing the Prompt Message 8-1
Creating and Using Abbreviations 8-4
Global Variables 8-10
Creating Login Files 8-13
Sending Messages 8-15
Disk Quotas 8-18

Part II: Programming

9 **Introduction to PRIMOS Programming** 9-1
For Experienced Programmers 9-1
Introductory Overview 9-1

10	**Compiling Programs**	10-1
	The Source File	10-1
	Invoking Compilers	10-2
	Object Files	10-4
	Addressing Modes	10-5
	Listing Files	10-6
	Cross-references	10-7
	Compiler Messages	10-7
	Combining Languages in a Program	10-8
11	**Linking Programs**	11-1
	Advantages of the BIND Linker	11-1
	The Linking Process	11-2
	Using BIND	11-3
	Using LOAD	11-8
12	**Running Programs Interactively**	12-1
	Using the Interactive Environment	12-1
	The RESUME Command	12-2
	The START Command	12-3
	Runtime Error Messages	12-5
	The Command Environment	12-5
13	**Debugging Programs**	13-1
	DBG Code Requirements	13-1
	The Features of DBG	13-2
	Compiling and Linking the Program	13-4
	Invoking and Terminating the Debugger	13-5
	Examining the Source Code	13-5
	Starting Program Execution	13-8
	Stopping Execution	13-8
	Examining and Modifying Data	13-10
	A Sample Debugging Session	13-11
	Part III: PRIMOS System Facilities	
14	**Command Files**	14-1
	Command Output Files	14-1
	Command Input Files	14-4
15	**The Basics of CPL**	15-1
	How Does CPL Work?	15-1
	Creating and Executing CPL Programs	15-2
	Using PRIMOS Commands in CPL Programs	15-3
	Using Variables in CPL Programs	15-4

	Decision Making (Branching) in CPL Programs	15-8
	Using Functions in CPL Programs	15-12
	Using CPL With Subsystems: &DATA Groups	15-13
	How CPL Programs End: The &RETURN Directive	15-15
	Errors in CPL Programs	15-16
	Debugging CPL Programs	15-16
	CPL Directives Summary	15-17
16	**Phantom and Batch Job Processing**	16-1
	Phantom Execution	16-1
	Batch Execution	16-5
17	**File-handling Utilities**	17-1
	Sorting Files	17-1
	Comparing Files	17-6
	Merging Text Files	17-8
	Joining Several Files Sequentially	17-10
18	**Tapes**	18-1
	Assigning Tape Drives	18-1
	Releasing Tape Drives	18-3
	Backing Up and Restoring Files	18-5
	Other Magnetic Tape Operations	18-9
19	**PRIMENET**	19-1
	Remote File Access	19-2
	Remote Login	19-7
	Using NETLINK for Remote Access	19-9
	Transferring Files Between Systems With FTS	19-13
20	**The Condition Mechanism**	20-1
	Using the Condition Mechanism	20-1
	The System Default On-unit	20-2
	Writing On-units	20-2
	Appendices	
A	**Glossary**	A-1
B	**Systems Defaults and Constants**	B-1
	Terminal Defaults	B-1
	PRIMOS Keyboard Standards	B-1
	PRIMOS Keyboard Defaults	B-1
	PRIMOS Command Line Standards	B-2
	Protection	B-2

C **The Prime Extended Character Set** C-1
Specifying Prime ECS Characters C-1
Special Meanings of Prime ECS Characters C-2
PRIME Extended Character Set Table C-2

D **Error Messages** D-1
Runtime Error Messages D-1
Batch Error Messages D-15

E **Ed Command Summary** E-1
ED Defaults E-5
ED Symbols E-6

F **Directory Passwords** F-1
Assigning Directory Passwords F-1
Using Passwords to Gain Access to Directories F-2
Setting Access Rights on File System Objects F-2
Converting a Password Directory to an ACL Directory F-3
Converting an ACL Directory to a Password Directory F-5
Creating a Password Subdirectory Under an ACL Directory F-6

G **System Information** G-1

Index Index-1

About This Book

The *PRIMOS User's Guide* provides an intermediate level introduction to the PRIMOS® operating system, including the file system, program development environment, and system facilities. The presentation shows you step by step how to carry out basic operations with the operating system and many subsystems. Many practical examples are included, and references to other Prime documents tell you where to find further information.

Part II of this book assumes that you have some programming background in a high level language. The remainder of the book requires no previous programming experience. The book assumes that you have no familiarity with Prime equipment or software.

Organization of This Book

The *PRIMOS User's Guide* has three parts.

Part I, PRIMOS BASICS, introduces the PRIMOS operating system, covering all of the basic operations you need to do practical work with the system.

- Chapter 1 introduces PRIMOS and describes basic operations, such as establishing a connection, logging in, and logging out. It also discusses PRIMOS command format and the mechanics of giving commands.
- Chapter 2 describes the structure of the PRIMOS file system.
- Chapter 3 introduces basic commands for working with the file system.
- Chapter 4 shows you how to create text files using the text editor ED, and how to print text files with the SPOOL command.
- Chapter 5 describes the PRIMOS file protection system, and introduces the commands you need to protect your files.
- Chapter 6 explains four enhancements to command line processing: iteration, wildcarding, treewalking, and name generation.
- Chapter 7 introduces the new PRIMOS Command Line Editor (ECL). ECL has features that let you edit command lines, recall and resubmit previous command lines, and quickly carry out repetitive operations.

- Chapter 8 shows you how to customize many aspects of your working environment. Topics include setting your own system prompts, defining abbreviations and global variables, creating command files, sending messages to other users, and setting file system quotas.

Part II, PROGRAMMING, introduces the basic features of the PRIMOS program development environment.

- Chapter 9 introduces the program development environment both to experienced and beginning programmers. The chapter briefly describes the steps of compiling, linking, running, and debugging programs in high-level languages, and introduces the PRIMOS subsystems involved in each step.
- Chapter 10 introduces the Prime language compilers, and shows you how to compile programs.
- Chapter 11 describes two of the Prime linking programs, BIND and LOAD. The chapter introduces Executable Program Formats (EPFs), the dynamic runfiles created by the BIND linker, and it shows you how to carry out basic linking operations with BIND and LOAD.
- Chapter 12 shows you how to run programs interactively at the terminal, and introduces the dynamic command environment made possible by EPFs.
- Chapter 13 provides an introduction to debugging programs with the Prime Source Level Debugger (DBG).

Part III, PRIMOS SYSTEM FACILITIES, describes a variety of PRIMOS system facilities and features.

- Chapter 14 describes command output files and command input files. Command output files allow you to capture terminal output in a file as you work with PRIMOS. Command input files let you provide PRIMOS command input from a file rather than from the terminal.
- Chapter 15 introduces the Prime Command Procedure Language (CPL) for carrying out complex operations involving PRIMOS commands. The chapter describes the basic language elements and shows you how to write simple CPL programs.
- Chapter 16 shows you two ways to run programs without tying up your terminal: phantoms and the Batch subsystem.
- Chapter 17 tells you how to use file-handling utilities to sort, compare, merge, and concatenate files.
- Chapter 18 explains how to use magnetic tapes on a Prime system.
- Chapter 19 explains the PRIMENET™ networking facility. Topics covered include remote file access, remote login, the NETLINK subsystem for communicating with both Prime and other vendors' remote systems, and the File Transfer Service (FTS) for transfering files between networked systems.
- Chapter 20 shows programmers how to use the PRIMOS condition mechanism to deal with errors and other program interruptions.

A series of appendices provides the following information:

- A glossary of terms used in Prime documentation
- A list of system defaults and constants
- The Prime Extended Character Set
- A list of system error messages
- A summary of ED commands
- A discussion of the password file protection system as an alternative to ACLs
- A table of commands that provide useful system information

Changes in This Edition

Parts I and II of the *PRIMOS User's Guide* have been substantially rewritten for this edition. Substantial changes have also been made to many of the chapters in Part III. Revisions reflect technical changes at Rev. 22.0, elimination of obsolete material, and rewriting to clarify or amplify existing information. Two completely new chapters have been added; Chapter 7 describes the new Command Line Editor ECL, and Chapter 9 provides a summary introduction to the PRIMOS program development environment.

Other Useful Books for New Users

For a beginner-level introduction to the PRIMOS operating system, see *Introduction to PRIMOS* (DOC10111-1XA). For more detailed information about the topics covered in this book, see the following:

- *PRIMOS Commands Reference Guide* (DOC3108-7LA) contains complete information on format and usage of all PRIMOS user commands.
- *New User's Guide to EDITOR and RUNOFF* (FDR3104-101A/101B) (change sheets UPD4033-21A and UPD4033-22A) and *EMACS Primer* (IDR6107-183P) give information about the two Prime supported text editors.
- *CPL User's Guide* (DOC4302-3LA) gives complete explanations, both for beginners and more advanced users, of the Prime Command Procedure Language.
- *User's Guide to Prime Network Services* (DOC10115-1LA) provides detailed information about Prime network facilities.
- *Advanced Programmer's Guide Series, Vols. 0-III* (DOC10066-3LA, DOC10055-1LA and update UPD10055-11A, DOC10056-2LA, DOC10057-1LA), gives detailed information about PRIMOS and programming for systems programmers.

Other books referred to in this book are

- *Subroutines Reference I: Using Subroutines* (DOC10080-2LA)
- *Subroutines Reference II: File System* (DOC10081-1LA) (updates UPD10081-11A and UPD10081-12A)

- *Subroutines Reference III: Operating System* (DOC10082-1LA) (updates UPD10082-11A and UPD10082-12A)
- *Subroutines Reference IV: Libraries and I/O* (DOC10083-1LA) (updates UPD10083-11A and UPD10083-12A)
- *Subroutines Reference V: Event Synchronization* (DOC10213-1LA)
- *Source Level Debugger User's Guide* (DOC4033-193L) (updates UPD4033-21A, UPD4033-22A, and UPD4033-23A)
- *EMACS Reference Guide* (DOC5026-2LA)
- *Programmer's Guide to BIND and EPFs* (DOC8691-1LA) (update UPD8691-11A)
- *SEG and LOAD Reference Guide* (DOC3524-192L) (update UPD3524-21A)
- *MAGNET User's Guide* (DOC10156-1LA) (update UPD10156-11A)
- *FORTRAN 77 Reference Guide* (DOC4029-5LA)
- *Data Backup and Recovery Guide* (DOC10129-1LA) (update UPD10129-11A)

For a complete listing of Prime documentation, see the *Guide to Prime User Documents* (DOC6138-6PA). Each listing in the guide includes a description of the book, its printing history, and its intended audience.

An up-to-date cumulative listing of manuals, updates, and programmer's companions is available online whenever you are logged in to PRIMOS by typing `HELP DOCUMENTS`.

Prime Documentation Conventions

The following conventions are used in command formats, statement formats, and in examples throughout this document. Examples illustrate the uses of these commands and statements in typical applications.

Convention	*Explanation*	*Example*
UPPERCASE	In command formats, words in uppercase bold indicate the names of commands, options, statements, and keywords. Enter them in either uppercase or lowercase.	**SLIST**
italic	In command formats, words in lowercase bold italic indicate variables for which you must substitute suitable values. In text and in messages, variables are in non-bold lowercase italic.	**LOGIN *user-id*** Supply a value for *x* between 1 and 10.
Abbreviations in format statements	If a command or option has an abbreviation, the abbreviation is printed in red.	**SET_QUOTA**

Convention	*Explanation*	*Example*
Brackets [ ]	Brackets enclose a list of one or more optional items. Choose none, one, or several of these items.	**LD** $\begin{bmatrix}\textbf{-BRIEF}\\ \textbf{-SIZE}\end{bmatrix}$
Braces { }	Braces enclose a list of items. Choose one and only one of these items.	**CLOSE** $\begin{Bmatrix}\textbf{\textit{filename}}\\ \textbf{ALL}\end{Bmatrix}$
Braces within brackets [{}]	Braces within brackets enclose a list of items. Choose either none or only one of these items; do not choose more than one	**BIND** $\left[\begin{Bmatrix}\textbf{\textit{pathname}}\\ \textbf{\textit{options}}\end{Bmatrix}\right]$
Vertical bars ‖ ‖	Vertical bars enclose a list of items. Choose one or more of these items.	**OUTPUT** $\begin{Vmatrix}\textbf{\textit{filename}}\\ \textbf{TTY}\end{Vmatrix}$
Parentheses ( )	In command or statement formats, you must enter parentheses exactly as shown.	**DIM** ***array (row, col)***

User input in examples	In examples, user input is printed in red but system prompts and output are not.	`OK, RESUME MY_PROG`
		`This is the output`
		`of MY_PROG.CPL`
		`OK,`

Ellipsis . . .	An ellipsis indicates that you have the option of entering several items of the same kind on the command line.	**SHUTDN** ***pdev-1*** **[...*pdev-n*]**
Hyphen –	Wherever a hyphen appears as the first character of an option, it is a required part of that option.	**SPOOL –LIST**
Subscript	A subscript after a number indicates that the number is not in base 10. For example, the subscript 8 is used for octal numbers.	200_8
Key symbol	In examples and text, the name of a key enclosed by a rectangle indicates that you press that key.	Press $\boxed{\text{Return}}$

Part I: PRIMOS Basics

1 Getting Started

You can use a Prime computer in many ways, but you use a few basic procedures every time you work at the terminal. This chapter explains the fundamental things you need to know to get started using a Prime computer. The topics covered include

- An overview of the PRIMOS operating system
- The terminal
- The PRIMOS command line
- Essential PRIMOS commands

Some details vary from system to system, but this chapter provides enough information so that any user can get started. Whenever important differences exist between systems, the text directs you to sources of more complete information.

Note

This chapter introduces the PRIMOS operating system to users with some previous computer experience. If you are a computer beginner, consult *Introduction to PRIMOS* first for an elementary level tutorial. Once you are familiar with the material covered in *Introduction to PRIMOS*, use the *PRIMOS User's Guide* as an intermediate level guide.

The PRIMOS Operating System

PRIMOS is the operating system used by all 50 Series™ computers. An **operating system** is a program that manages the hardware and software resources available in a computer system. PRIMOS has three basic tasks:

- PRIMOS controls hardware. PRIMOS acts as an insulating layer between users and the mechanics of devices, such as the central processing unit, memory, storage devices (such as disk drives), and input and output devices (such as terminals and printers). For example, when you use a text editor program to create and store a document on disk, neither you nor the text editor needs to worry about the actual mechanics of putting the data on the disk. When you ask the editor to save your document, the editor in turn asks the operating system to direct the complex hardware operations required.

- PRIMOS manages software. PRIMOS runs programs for users and organizes data in the PRIMOS file system. When you want to run a program, for example, PRIMOS takes care of fetching the program from storage, allocating memory space for it, and initiating its operation. When you save a file on disk, the file becomes part of the PRIMOS file system so that you can easily retrieve it for further use.
- PRIMOS shares system resources among users. PRIMOS manages both hardware and software resources so that different users and tasks can share them efficiently and without interference. For example, PRIMOS provides each user with a private memory space that no other user has access to. Much of the time PRIMOS allows you to work with the computer as if it were yours alone, even though many other users may be working at the same time. Depending on the central processing unit, PRIMOS can support as many as 960 simultaneous users.

Three features of PRIMOS are especially important to users:

- The user interface
- Compatibility
- Network connections

The User Interface

You ask PRIMOS to carry out specific tasks by giving commands at the terminal. The PRIMOS user interface is flexible; it allows you to give commands and carry out work both interactively and non-interactively.

Using PRIMOS Interactively: You communicate with PRIMOS principally by means of an **interactive dialog** carried out at your terminal. You type one or more commands to PRIMOS, and PRIMOS immediately attempts to carry them out. PRIMOS informs you when it completes each task or lets you know that problems have arisen. Many commands also cause PRIMOS to send further output to your terminal. If you ask PRIMOS to list the names of some files, for example, PRIMOS responds by writing the list to your terminal screen. The process is interactive because you can tailor your commands to the responses from PRIMOS.

Other Ways to Use PRIMOS: Sometimes you may find it convenient to use PRIMOS with little or no terminal interaction. PRIMOS provides several ways to do this:

- **Command input files** and **Command Procedure Language** (CPL), let you store a series of PRIMOS commands in a file. With CPL you can also include special directives to control command execution. You can use command input files and CPL to save frequently used command sequences and submit them to PRIMOS for execution when you need them.
- **Batch processing** allows you to submit a series of commands that the computer executes when it has the time and resources to carry them out efficiently. After you submit them, Batch jobs run without terminal interaction, so you can do other work while you wait for them to finish.

- **Phantom processes** run immediately without terminal interaction. You can run programs as phantoms and then use the terminal for other work while they are running.

Most work with PRIMOS is interactive. In fact, you need to give PRIMOS commands interactively in order to use any of the non-interactive procedures.

Compatibility

PRIMOS is designed specifically to work with 50 Series computers, and the 50 Series hardware is tailored to run PRIMOS with maximum efficiency. You use exactly the same operating system commands and features on all 50 Series machines. You can run programs created on one 50 Series computer on other 50 Series machines with little or no modification. Upward compatibility is complete; you can always run software created on one 50 Series machine on other machines with equal or higher memory capacity. Downward compatibility is also high; you can run software on machines with smaller memories as long as you observe constraints on program size and compilation mode.

Networks

PRIMOS users can have access to the resources of more than one computer when computers are linked together with Prime network hardware and software. PRIMOS supports a variety of local and wide area networks.

Local Area Networks: **Local Area Networks** (**LANs**) link nearby computers together over direct lines. Prime computers can be connected together in three types of LANs, illustrated in Figure 1-1.

- The RINGNET™ network connects several computers together in a ring configuration. Each user terminal is connected directly to one of the computers on the ring.
- The Prime LAN300 connects both user terminals and computers to a single shared communication line.
- Point-to-point connections can tie together networks as well as single computers.

In each case, PRIMENET™ software gives you **transparent** access to disk files throughout the network. You can use files on other machines on the network in the same way you use files located on the machine you are directly connected to. PRIMENET and other Prime software also allow you to log in to more than one machine in the network.

Wide Area Networks: A variety of Prime hardware and software also gives users access to the resources of computers linked in wide area networks. **Wide area networks** connect machines over long distances using public communication lines. Prime network hardware and software support widely accepted communications standards so that PRIMOS users can transfer files among computers and work remotely on both Prime machines and those of other vendors.

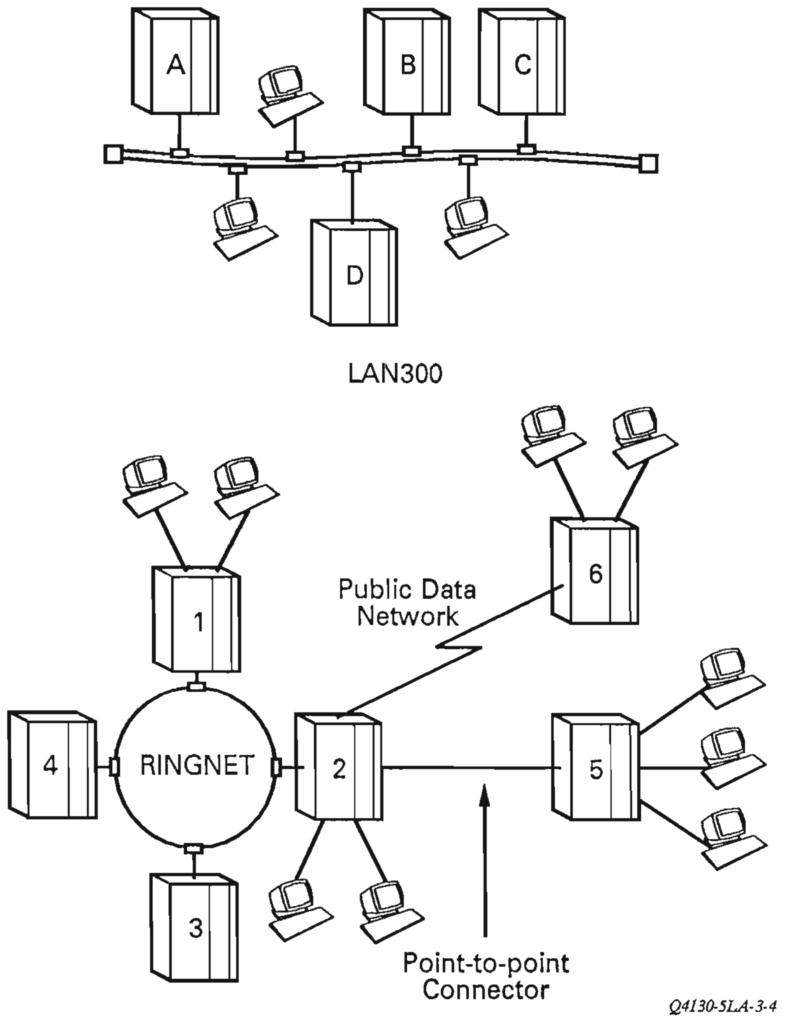


FIGURE 1-1
Prime Local Area Networks

The Terminal

You communicate with PRIMOS using the terminal. You give commands by typing them at the keyboard, and PRIMOS responds by putting its output on the terminal screen. This section tells you how to establish a connection between your terminal and the computer.

Note

Terminals vary a great deal from model to model. The discussion that follows assumes that you have turned on your terminal and are familiar with the locations of the terminal keys. If you are not familiar with the basic operations of your terminal, consult the terminal's documentation. *Introduction to PRIMOS* includes a general introduction to terminal use for users with little or no experience.

Connections

Before you can start working at your terminal, you may need to establish a communication link between the terminal and the computer. If you have a direct connection, you are in communication with the computer whenever the terminal is turned on. If you have a dialup (telephone) connection, or if your terminal is connected through the Prime LAN300 local area network, you need to take certain preliminary steps to establish a communication link.

Dialup Terminals: If you are using a dialup line, consult with your System Administrator about the procedures required to establish a communications link. On some systems you merely need to dial in. On others you may, for security reasons, need to go through a more elaborate procedure. You may also have to set your terminal's communication rate, parity, and other characteristics to match those of the computer you are connecting with. If your system has **Auto Speed Detect**, the computer can match your terminal's communication rate automatically if you press $\boxed{\text{Return}}$ several times after establishing the dialup connection. For further information on remote connections, consult the *User's Guide to Prime Network Services*.

LAN300: If you are using LAN300, your terminal is connected through hardware called the **LAN Terminal Server**, rather than directly to a specific computer. **Network Terminal Service** (NTS) software controls your communication with computers on the network. You need to tell NTS to connect your terminal to one of the computers on the network before you can begin to work. Details of this process are noted in the sections, The Command Line, and Logging In With LOGIN, later in this chapter. The first time you begin to work on a terminal connected through NTS, you may need to press $\boxed{\text{Return}}$ several times before you get any response from NTS. This procedure allows NTS to adjust its communication rate to match the characteristics of your terminal and to store the information for future terminal sessions.

The User's Dialog With PRIMOS

Once you have established a connection with the computer, your dialog with PRIMOS consists of two basic elements: **prompts** and **commands**. This sections discusses prompts. The next section shows you how to give commands with the PRIMOS command line.

Prompts

A prompt is a message that PRIMOS puts on the screen to indicate that it is ready to receive commands. PRIMOS displays two standard prompts:

Prompt	*Meaning*
`OK,`	Indicates that the most recent PRIMOS command has been successfully executed, and that PRIMOS is ready to accept another command.
`ER!`	Indicates that PRIMOS was for some reason unable to execute the most recent command and is now ready to accept another command from the user. The `ER!` prompt is usually preceded by one or more error messages indicating the nature of the error. These messages can be cryptic; consult Appendix D for explanations. Many errors are simply typographical. Others result from either leaving out command arguments (discussed in the next section) or incorrectly specifying pathnames (discussed in Chapter 2).

Note

The `OK,` and `ER!` prompts are the PRIMOS **default** prompts. They are the prompts PRIMOS uses unless you specify alternate forms. Many aspects of the PRIMOS user environment have default values that you can change. Explanations of how to change various defaults are given in this chapter in the section, Setting Terminal Characteristics With TERM, and in Chapter 8, Customizing Your Environment.

Prompts are an important element in the user dialog with PRIMOS, because they let you know when PRIMOS is ready to receive commands from the terminal. While PRIMOS is executing other interactive commands, it can't accept commands from the terminal. A prompt doesn't appear until PRIMOS is ready to accept another command.

Type-ahead: If you type commands before a prompt has appeared, PRIMOS simply stores your input and then processes it when it is ready to accept further commands. This feature is called **type-ahead**. Because each character **echoes** (appears on the screen) as you type it, output from previous commands may become mixed with the commands being typed ahead. In these cases PRIMOS can still interpret your commands correctly, but the screen display may be difficult to read. Type-ahead is limited by the size of the terminal **input buffer**, an area of memory that holds characters typed at the terminal. The terminal input buffer has a default capacity of 192 characters.

Note

If you use the Command Line Editor (ECL), PRIMOS redisplays the typed ahead command line after the next prompt, just as if you had waited for the prompt before typing the command line. See Chapter 7, Command Line Editor.

System and Subsystem Prompts

Prompts are useful signposts as you work with the PRIMOS operating system. Many of the programs that run under PRIMOS are also interactive; they also put prompts on the screen and expect commands from the user. In general, neither the prompts nor the commands are the same as the PRIMOS prompts and commands. For example, if you are using the text editor program ED, the error prompt is a question mark (`?`). This distinction is useful, because it helps you keep track of where you are in the system. The form of the prompts lets you know whether you are communicating directly with PRIMOS, so that you can give PRIMOS commands, or are communicating with an interactive program that requires different commands.

Because they indicate that you are communicating directly with the operating system, the PRIMOS prompts are called **system prompts** to distinguish them from the prompts used by programs that run under PRIMOS. The `OK,` and `ER!` prompts (or whatever you have changed them to) are signals from a section of the PRIMOS operating system called the **command processor** that it is ready to process a user command. When you are communicating directly with PRIMOS via the command processor, you are said to be at PRIMOS **command level** or **system level**. Interactive programs such as ED that run under PRIMOS are often called **subsystems**. The different prompts thus indicate whether you are at command level or in a subsystem.

The Command Line

Once you receive a system prompt, you communicate with PRIMOS by entering a **command line** at the terminal. This section describes command line format and shows you how to use special keys and characters to modify command lines and carry out other functions.

Command Line Format

A command line consists of one or more PRIMOS commands and any appropriate **arguments, options**, symbols, and punctuation. The format of the PRIMOS command line is the same for all commands:

COMMAND [***argument*** [...***argument***]] [**–OPTION** ***argument*** [...**–OPTION** ***argument***]]

Term	*Meaning*

COMMAND	Specifies a PRIMOS command. Command names often have both long and short forms, either of which can be used in the command line. In this book, the short form is indicated by a portion of the long form given in red text. For example, the command
	
	LOGOUT
	
	can be typed either as
	
	`OK, LOGOUT`
	
	or as
	
	`OK, LO`

argument	Identifies an element to be acted upon by the command. Arguments are usually the pathnames of files or directories (explained in Chapter 2) or identifying names such as a user ID (explained below). Not all commands take arguments, and some commands can be given with or without arguments.

–OPTION	Specifies an optional term that modifies the action of the command. Not all commands take options; a list of possible options is documented with each command that takes them. Some options also take arguments. Option names always begin with a hyphen (–).

Not all elements are present in every command line, but the order of elements is always the same: the command name, followed by any arguments, followed by any options.

You can place several commands on a single command line, separated by semicolons:

COMMAND [*arg*] [**–OPTION**] ; **COMMAND** [*arg*] [**–OPTION**] ; ... **COMMAND** ...

PRIMOS accepts command lines with a maximum of 160 characters.

You must type commands either in their full or abbreviated forms exactly as they are shown in the command descriptions. You must end all command lines by typing $\boxed{\text{Return}}$. $\boxed{\text{Return}}$ may also be called $\boxed{\text{CR}}$ (Carriage Return), or $\boxed{\text{Enter}}$ on some terminals.

Modifying and Editing Command Lines

You can modify a command line at any point before you type $\boxed{\text{Return}}$, because PRIMOS doesn't begin to process your command line until you type $\boxed{\text{Return}}$. You can modify command lines in two ways: with the PRIMOS erase and kill characters, or with the command line editor.

PRIMOS Erase and Kill Characters: PRIMOS defines two characters that you can use to make simple modifications to command lines before you type $\boxed{\text{Return}}$.

Character	*Meaning*
$\boxed{"}$	The default **erase character.** $\boxed{"}$ does not actually erase any characters from your screen, but when PRIMOS processes the command line, each $\boxed{"}$ removes a preceding character from the current line. Erasure is from right (the most recent character) to left. Two $\boxed{"}$s erase two characters, three erase three, and so forth. The PRIMOS command TERM (described in the next section) allows you to choose a different erase character, such as $\boxed{\text{Backspace}}$, in order to make corrections more convenient and easier to read.
$\boxed{?}$	The default **kill** character. Each $\boxed{?}$ removes all preceding characters on the line when PRIMOS processes the command line. The kill character can also be modified with the TERM command.

You can use these erase and kill characters both at PRIMOS command (system) level and in many subsystems. The prompts and commands are different, but the mechanics of typing and modifying commands are essentially the same at all levels of the system. When you give commands to ED, for example, the $\boxed{\text{Return}}$, erase, and kill characters work exactly as they do at PRIMOS command level.

The Command-line Editor: The default procedures are adequate for editing PRIMOS command lines when you give short, simple commands. With long command lines or many repetitions of the same or similar commands, you may want to use the PRIMOS commmand-line editor, EDIT_CMD_LINE (ECL). You use ECL at PRIMOS command level to modify, delete, and insert text in the current command line as well as to recall, modify, and reissue previously entered command lines. The features of ECL are not yet available in most subsystems. ECL is documented in Chapter 7.

Control Key Characters

You normally communicate with PRIMOS by means of the command line after a prompt. Sometimes you need to communicate with PRIMOS before it has completed its current task and given you a prompt. Three special key combinations serve this purpose. To use them, press the Control key Ctrl *simultaneously* with another keyboard character.

Keys	*Function*
Ctrl P	Immediately halts whatever command is currently executing and returns you to PRIMOS command level. `QUIT` appears on your screen. Ctrl P allows you to escape from a command or program that is not functioning as desired, although it may leave files open and have other undesirable effects. Commands to clean up after a command is halted with Ctrl P are given in Chapter 3. If the Break key is enabled at your terminal, it may also function like Ctrl P.
Ctrl S	Halts output to the terminal. This is useful when terminal output longer than a single screen is **scrolling** (disappearing off the top of the screen) faster than you can read it. In order to function, Ctrl S must be activated by the TERM –XOFF command documented in the next section.
Ctrl Q	Resumes output to the terminal following a Ctrl S if these functions have been activated with TERM –XOFF.

These control key combinations function both at the PRIMOS command level and in many subsystems.

PRIMOS Reserved Characters

PRIMOS reserves a set of keyboard characters for special uses. Do not use them in command lines except as specifically documented:

```
, ( ) { } [ ] < > ! % ' = + ` ~ : | ; ? " \ ^
```

Special Characters in Subsystems

Some subsystems define their own special characters for specific purposes. Such special characters are documented along with the subsystems that use them. Some programs running under PRIMOS may disable special characters or use them for other purposes. The EMACS text editor, for example, responds to Ctrl P by asking if you really want to halt the program. This feature prevents you from inadvertently losing text if you accidently type Ctrl P while using EMACS.

Essential PRIMOS Commands

You need to know a few basic commands in order to start working with PRIMOS:

Command	*Function*
LOGIN	Identifies you as an authorized user of the system and allows you to start work.
LOGOUT	Protects your work, closes your files, and frees up space for other users when you finish working.
CHANGE_PASSWORD	Allows you to select a secret password to protect your work.
TERM	Sets terminal characteristics, allowing you to customize many aspects of your working environment.
HELP	Gives information on PRIMOS commands.

Logging In With LOGIN

In order to start working with PRIMOS you have to log in. Logging in identifies you to PRIMOS as an authorized user. When you log in, PRIMOS sets aside system resources for you and gives you access to the system. You log in with the LOGIN command. Until you have successfully logged in, the command processor cannot process any other PRIMOS commands you give.

User IDs and Passwords: In order to log in, you need a user ID and possibly a password. Your System Administrator issues you a user ID and, if necessary, a password before you log in for the first time.

Note

The **System Administrator** is a person designated to control may aspects of your system's operation. One of the System Administrator's tasks is to establish new users on the system.

The **user ID** is the name by which you are known to your computer system. User IDs can have a maximum of 32 characters. The first character of the ID must be a letter, and the rest may be any combination of letters, digits, periods, underscores, and dollar signs. Case does not matter, because PRIMOS converts lowercase letters to uppercase.

Your system may require a **login password** to prevent unauthorized access. It may consist of a maximum of sixteen ASCII characters (listed in Appendix C). It may not include any of the PRIMOS special keys and reserved characters documented in this chapter. The System Administrator supplies your password before you log in for the first time. You can change this password to another one after you log in.

LOGIN Command Format: The simplest format for the LOGIN command is

```
LOGIN
```

After you enter this command, the system requests a user ID with the prompt

```
User id?
```

Type your user ID. If you have a login password, the system requests it with the prompt

```
Password?
```

Type your login password. For security reasons, the password does not appear on the screen as you type it.

Computer Generated Passwords: Some systems may provide **computer generated passwords**. If this is the case, you still receive a password from the System Administrator, but you automatically receive a new computer generated password when you log in for the first time. When you first log in, you see the following message:

```
Computer generated passwords are in effect.
Please ensure that you can view your new password in privacy.
Type RETURN to continue:
```

After you type Return, the computer generates a new password for you. For example,

```
Your new password is BAGINUC
Reenter new password for confirmation:             Does not appear on screen.
Your new password has been confirmed.
```

Password Lifetime: Your System Administrator may choose to limit the lifetime of your password. If this is the case, you need a new password when the old one expires. When your password has expired, the system notifies you after you log in.

On systems that allow users to choose their own passwords, you can choose any new password. The following example shows how the system notifies you that your password has expired and prompts you for a new one:

```
LOGIN
User ID?
YOURID
Password:                                          Type your password as usual.
Your password has expired; please change it.
New password:                                           Type a new password.
                                                   It does not appear on screen.
Reenter new password for confirmation:             Type the new password again.
Your new password has been confirmed.
```

On systems that provide computer generated passwords, the procedure is as follows:

```
LOGIN
User id?
YOURID
Password:                                          Type your password as usual.
Your password has expired.

Computer generated passwords are in effect.
Please ensure that you can view your password in privacy.
Type RETURN continue:
Return
```

```
Your new password is NOMANES.
Reenter new password for confirmation: The password does not appear on screen.
Your new password has been confirmed.
```

Project IDs: Some systems organize users into specific groups called **projects**. If your system uses the project structure, PRIMOS may now request a project ID with the prompt

```
Project ID?
```

Type in your project ID. If you receive a project ID prompt and you do not have a project ID, see your System Administrator.

You must always provide the system with your user ID. Whether you must supply a password or a project ID depends upon your installation. Once you have successfully provided all of the information requested, the login procedure is completed, and PRIMOS responds as in the following example:

```
LOGIN
User id? FRED
Password?                                   The password does not appear on screen.
Project id? RESEARCH

FRED (user 13) logged in Wednesday, 14 Dec 88 11:23:28.
Welcome to PRIMOS version 20.0
Copyright (c) 1988, Prime Computer Inc.
Last login Monday, 12 Dec 88 09:49:44.
```

RESEARCH is the project ID. The number in parentheses is a user number assigned by PRIMOS. The time is expressed in 24-hour format (*hh:mm:ss*).

Problems Logging In: If you misspell your user ID or password, you receive the message

```
Invalid user id or password; please try again.
```

If you enter a project ID incorrectly, you receive the message

```
Invalid project id; please try again.
```

If you or someone else has tried and failed to log in using your user ID, you see a warning after you log in successfully:

```
Warning!  There were 3 failed attempts to log in under this id since
          the last successful login.
```

This is a security measure, designed to let you know if an unauthorized person has attempted to log in with your user ID.

If you repeat the login process and still have trouble, ask your System Administrator for help. If the system is already being used to capacity, a message such as `maximum number of users exceeded` may be displayed. In this case, try to log in again later, when some other user may have logged out.

Alternate Form of the LOGIN Command: You may also enter arguments and options on the same line as the LOGIN command (although for security reasons your System Administrator may disallow passwords on the login line). The format is

LOGIN [*user-id* [*login-password*]] [-PROJECT *project-id*]

Argument/Option	*Description*
user-id	Your user identification name.
login-password	Your password may be included on the login line if your system allows it.
-PROJECT ***project-id***	Your project ID associates you with a particular project.

If you give your login password on the command line, you must also include your user ID. For example,

```
LOGIN FRED NIX

FRED (user 13) logged in Wednesday, 14 Dec 88 12:27:21.
Welcome to PRIMOS version 20.0
Copyright (c) 1988, Prime Computer Inc.
Serial #
Last login Monday, 12 Dec 88 11:23:28.
```

Logging In With LAN300: If your terminal is connected to the Prime LAN300 local area network, you need to establish a connection with one of the computers on the network before you can log in. You do this by giving the NTS CONNECT command after you switch your terminal on. To use the CONNECT command, you need to know the name of the system you are logging in to. If you don't know, ask your System Administrator.

The following example shows how to use the NTS CONNECT command. Suppose you want to log in to a system called SYSA. After turning on your terminal, you see the NTS prompt:

```
CMD1
```

If the prompt doesn't appear right away, you may need to press Return several times. Once you see the prompt, type

```
CMD1 CONNECT SYSA
```

to establish a connection. The system responds with

```
SYSA CONNECTED
```

Now you can log in using the PRIMOS LOGIN command, as explained above.

Assigning or Changing Your Login Password With CHANGE_PASSWORD

After you log in, you can change your current login password with the CHANGE_PASSWORD command. The format is

CHANGE_PASSWORD ***old-login-password***

On systems that allow users to choose their own passwords, the system requests the new login password with the prompt

```
New password?
```

As usual, the new password does not appear on the terminal as you type it. The system requests the new login password a second time, for verification, with the prompt

```
Reenter new password for confirmation:
```

PRIMOS displays an appropriate error message if the old password is incorrect, the two new passwords entered do not match, or the format of the new password is illegal. In any of these cases, the old password is not changed.

An example of CHANGE_PASSWORD follows:

```
OK, CHANGE_PASSWORD NIX                        The old password is NIX.
New password?                       New password does not appear on screen.
Reenter new password for confirmation:
OK,
```

On systems that use computer generated passwords, CHANGE_PASSWORD causes the computer to generate a new password for you. For example,

```
OK, CPW

Computer generated passwords are in effect.
Please ensure that you can view your password in privacy.
Type RETURN continue:
Return
Your new password is CUNUJEH.
Reenter new password for confirmation: The password does not appear on screen.
Your new password has been confirmed.
```

Completing a Work Session With LOGOUT

When you finish a session at the terminal, give the LOGOUT command. The format is

LOGOUT

PRIMOS acknowledges the command with the following message:

```
user-id (user number) logged out weekday, date time.
Time used:  xxh xxm connect, xxm xxs CPU, xxm xxs I/O
```

The various parts of this display have the following meanings:

Term	*Description*
`user` *number*	The number assigned to you at LOGIN
time	Time of day expressed in 24-hour format (*hh:mm:ss*)
*xx*h *xx*m `connect`	The amount of elapsed clock time between LOGIN and LOGOUT, in hours and minutes
*xx*m *xx*s `CPU`	Central processing unit time consumed, in minutes and seconds
*xx*m *xx*s `I/O`	The amount of input/output time used, in minutes and seconds

For example,

```
LO

FRED (user 13) logged out Wednesday, 14 Dec 88 13:52:20.
Time used: 01h 22m connect, 01m 14s CPU, 03m 35s I/O.
```

It is good practice to log out after every session. Logging out closes all files and makes room on the system for another user. Logging out also protects your files and directories against unauthorized access by someone who comes across an unattended, but logged in, terminal.

However, if you forget to log out, the system automatically logs out your unused terminal after a time delay. This delay is set by the System Administrator. The default is 1000 minutes, but most System Administrators lower this value.

Setting Terminal Characteristics With TERM

You can modify several important aspects of your working environment by changing terminal characteristics with the TERM command. These characteristics remain in effect until you reset them or until you log out. The commonly used TERM options are listed below. Typing TERM with no options causes PRIMOS to print a list of all available TERM options. The format is

TERM ***options***

The common options are

Option	*Function*
–ERASE ***character***	Replaces the default " erase character with one chosen by the user. *character* is the new erase character.
–KILL ***character***	Replaces the default ? kill character with one chosen by the user. *character* is the new kill character.
–BREAK {ON / OFF}	Enables or disables use of Ctrl P as a break character, to interrupt a running program or command. The default setting, normally enabled at login, is –BREAK ON.
–XOFF	Enables the XOFF/XON feature, which allows users to suspend and resume terminal output with Ctrl S and Ctrl Q. Also sets the terminal to full-duplex to activate echo.

–NOXOFF	Disables the XOFF/XON feature and sets terminal to full-duplex. This is the default, normally enabled at login.
–DISPLAY	Displays currently set erase and kill characters, duplex, break, and XON/XOFF status.

For example, if you want to change the erase character from double quote to backslash, give the following command:

```
OK, TERM -ERASE \
```

Many users change the erase and kill characters to nonprinting characters such as Backspace. This makes corrections more readable, because Backspace and other nonprinting keys can actually move the cursor so that you can type over the erroneous part of the command line. To make this change, type the TERM command in the normal way, pressing the chosen key after the word –ERASE or –KILL. For example, to change the erase character to Backspace type

```
OK, TERM -ERASE Backspace
```

Backspace does not appear on the screen when you type it, but PRIMOS recognizes it as the new erase character.

Other TERM options are discussed in the *PRIMOS Commands Reference Guide*.

Requesting Information With HELP

If you need information about PRIMOS while you are logged in, you can often get it online with the HELP command. HELP gives you access to information about PRIMOS commands, groups of commands, and other PRIMOS topics. Its format is

HELP [*name*]

name is the name of the command or topic on which you wish information. *name* may be a command abbreviation. PRIMOS responds by printing a discussion of the command or topic you have chosen. References to further information and the date that the HELP information was written or updated appear at the end of the discussion. If you omit *name*, PRIMOS displays a list of available commands and topics with a descriptive phrase for each.

For example,

```
OK, HELP ORIGIN
ORIGIN                                Returns to origin directory

Abbreviation:  OR

    ORIGIN

The ORIGIN command changes the user's attach point to his origin

directory (initial attach point).  It takes no arguments.

For further information, see the Primos User's Guide.

October 1988
OK,
```

2

The PRIMOS File System

PRIMOS organizes information stored on disk in the PRIMOS file system. The file system is designed to make it easy to store, retrieve, and modify information while protecting it from accidental damage. This chapter explains the basic structure of the file system.

Note

Subsequent chapters describe the most important PRIMOS commands for working with the file system, show you how to create and print text files, and explain how you can protect file system objects. Taken together, Chapters 2 through 5 provide a detailed introduction to the PRIMOS file system. For a technical description of the file system, see the *Advanced Programmer's Guide, Vol. 3: The File System.*

File System Objects

The basic unit of the PRIMOS file system is the **file system object**. A file system object is a collection of information identified by an **objectname**. File system objects are classified into four types according to their function within the PRIMOS file system:

- File
- Directory
- Access category
- Segment directory

In practice, these types fall into two broad categories.

Files

A PRIMOS file is an organized collection of information identified by a **filename**. The contents of a file can be anything that a computer is capable of storing: text, such as documents or source programs; binary information, such as compiled object files; and all kinds of numeric and alphabetic information, such as the files created by database management systems. Much of the information in this book deals with commands and subsystems that create, modify, or process files of various types.

Directories, Access Categories, and Segment Directories

These special files contain information related to the organization and use of the file system itself.

Directories: A directory is a special file that contains a list of names of other file system objects, information about their characteristics (such as date and time last modified), and information that PRIMOS can use to locate them. In effect, a directory functions as an index to a group of file system objects.

Directories can also contain information about other directories so that the file system has a hierarchical structure. One directory can list several more directories, each of which can list further directories and files, and so on. This organization of the PRIMOS file system, called a tree structure, is further discussed in the next section.

Access Categories: An access category is a special file that PRIMOS uses to determine what access rights a user or group of users has to a given file system object or set of objects in the PRIMOS file system. Your **access rights** to an object determine the kinds of operations, such as reading, modifying, or deleting, that PRIMOS allows you to carry out on the object. If your System Administrator permits, you can set up access categories to protect your own files. Access categories and the PRIMOS file protection system are discussed in detail in Chapter 5.

Segment Directories: A segment directory is a specialized directory that the PRIMOS SEG utility uses to store information about segmented binary code. Some data management systems also use segment directories to organize information. Segment directories are not discussed in this book. See the *Advanced Programmer's Guide, Vol. II* and the *LOAD and SEG Reference Guide* for more information.

Figure 2-1 shows the symbols used in this book to illustrate the file system objects.

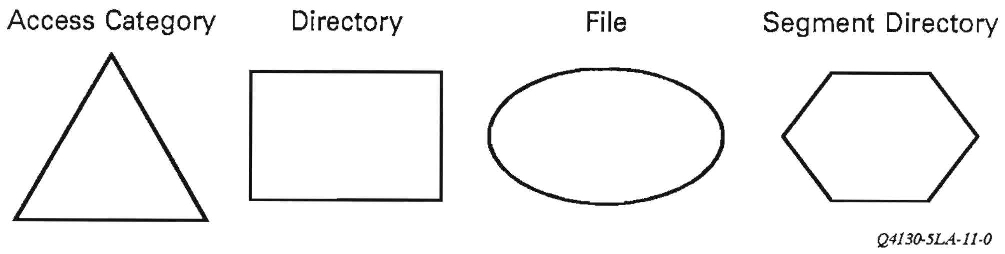


FIGURE 2-1
File System Object Symbols

Storage of File System Objects

The PRIMOS file system provides a conceptual structure for storing, retrieving, and manipulating file system objects so that you normally don't need to know about the physical details of storage. PRIMOS stores file system objects on magnetic disks and brings them into memory when they are needed. The file system associates every object with an **objectname** so that you can manipulate objects using their names without having to know anything about the physical locations of the objects. The major exception to this rule occurs when files are copied to magnetic tape for backup or long term storage. If you wish to use a file on tape, then you, the computer operator, or the System Administrator needs to know which tape the file is on, and must choose a tape drive to read the tape. Tape operations are discussed in Chapter 18.

PRIMOS stores files in a variety of formats, called **file types**. Differences among file types are not generally visible to users carrying out operations documented in this book. File types are discussed in the *Advanced Programmer's Guide, Vol. II*.

Naming File System Objects

Characters in an Objectname

Each file system object (file, directory, segment directory, or access category) is identified by an **objectname** (sometimes called an **entryname**) . An objectname may be from 1 through 32 characters in length, and may contain only the following characters:

A through Z
0 through 9
_ # $ - . * & /

On some devices underscore (_) prints as a backarrow (←). You can type objectnames using either uppercase or lowercase, but PRIMOS always converts them to uppercase. The first character of an objectname cannot be a digit. Avoid objectnames that begin with

_ & $ - .

as well as the objectname * because these characters have special meaning for some commands and subsystems and may cause confusion if used arbitrarily.

The Objectname Suffix

Objectnames, especially filenames, are customarily divided into two components: a **basename** and a **suffix**. The two components are separated by a period (.) in the objectname. The following examples show legal PRIMOS objectnames:

MYFILE.MSS
PROGRAM.FTN
SAMPLE2.RUN
LETTERS.AUGUST
MANUSCRIPTS

The first four examples are divided into basename and suffix. In these examples .MSS, .FTN, .RUN, and .AUGUST are the suffixes. An objectname may contain as many as 16 components, separated by periods. However, only the final component is considered to be the suffix. Names with more than three components are not recommended.

Suffix Conventions

When choosing objectnames, use a basename that gives some useful information about the contents of the object, and use a standard suffix that identifies the object's function. Some Prime programs that operate on file system objects create or expect to find objects with specific suffixes. Commonly used objectname suffixes that are recognized by Prime software include the following:

Suffix	*Meaning*
.compiler-name	Program source file. For example, a FORTRAN source file uses the filename suffix .FTN. A full list of compiler-name suffixes is given in Chapter 10.
.LIST	Listing file created by a compiler.
.RUN	Runfile created by BIND.
.SAVE	Runfile created by LOAD.
.SEG	Segment directory created by SEG.
.CPL	CPL file.
.ACAT	Access category.

Other common user suffixes, not recognized by Prime software but recommended to make your file organization easy to understand, include

Suffix	*Meaning*
.ABBREV	Abbreviation file.
.COMI	Command input file.
.COMO	Command output file.
.GVAR	Global variable file.
.PH	Phantom command file.
.RUNI	RUNOFF source (input) file.
.RUNO	RUNOFF output file.
.T	Temporary file.

You can differentiate groups of related objects by their filename suffixes. For example, if you write a FORTRAN program called PROGRAM1, the source file is normally called PROGRAM1.FTN; the binary file produced by the compiler, PROGRAM1.BIN; and the executable file, PROGRAM1.RUN or PROGRAM1.SAVE. In cases where no standard suffix exists, you can choose a suffix to distinguish an object from others with similar names. For example, you could use the names

```
LETTERS.AUGUST
LETTERS.SEPTEMBER
```

for two directories listing files of letters.

These naming conventions help you to keep track of the contents of objects in your directories. They also allow you to access groups of related objects by using wildcard characters (explained in Chapter 6).

The File System Tree Structure

PRIMOS organizes file system objects in a hierarchy called a **tree structure**. In the tree structure, directories can list the names of other directories, which can in turn list the names of further directories, and so on. This creates a branching structure like a tree. Figure 2-2 shows an example of a file system tree.

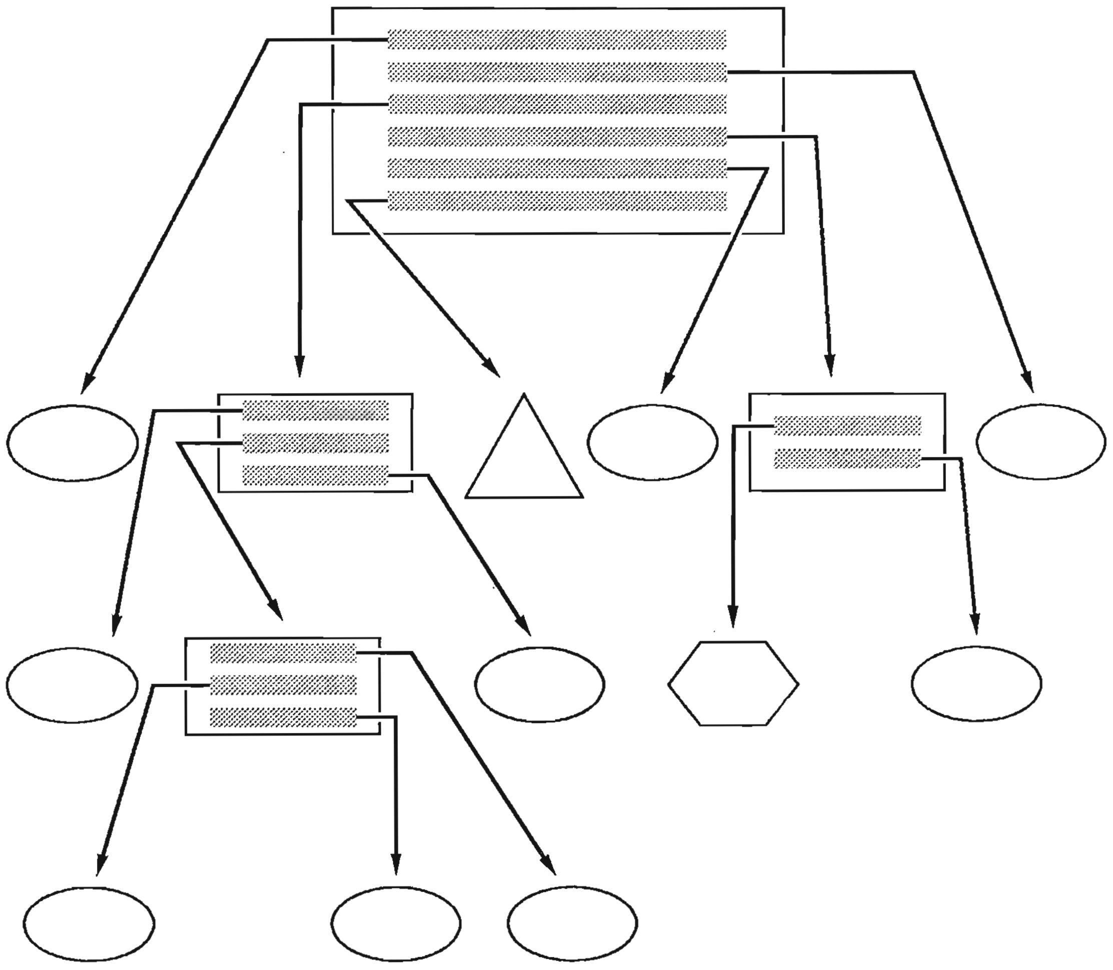


FIGURE 2-2
A PRIMOS File System Tree

PRIMOS organizes this structure with three classes of directories, each of which occupies a different position in the hierarchy.

- **Master File Directories (MFDs)** are at the highest level. PRIMOS divides available disk storage space into **logical disks** (also called **disks, partitions**, or **disk volumes**), and each logical disk has an associated MFD that lists the volume's contents. A logical disk is assigned to one or more physical disk surfaces on a disk storage device, but the file system treats a logical disk as a unit and associates it with a single MFD.
- **Top-level directories** occupy the next level. Usually, the System Administrator assigns a top-level directory to each user. Often projects and groups of users have their own top-level directories as well. Other top-level directories are used for system software. The MFD lists all of the top-level directories on a given logical disk.
- **Subdirectories** are directories created by users within top-level directories. Since subdirectories may contain further subdirectories, users can extend the hierarchical structure to any useful level.

Users with sufficient access rights can create file system objects of all types in their top-level directories and subdirectories: files, access categories, segment directories (created by the SEG utility), and subdirectories. Normally users do not have the right to create objects within an MFD.

A directory that immediately contains a subdirectory is called the **parent directory** of that subdirectory.

Pathnames

PRIMOS locates objects within the file system tree structure using pathnames. A **pathname** specifies a path through a series of directories and subdirectories to the object in question. An **absolute pathname** is one that begins with a disk name, followed by a top-level directory name and the names of any subdirectories, and ends with the name of the target object.

In many circumstances PRIMOS can find an object using a pathname that does not start with a disk name. The section, Shortening Pathnames, shows how to create such partial pathnames.

Specifying a Pathname

In an absolute pathname, angle brackets (<>) enclose the disk name. Right angle-brackets (>) separate the other objectnames from one another. For example,

```
<FOREST>BEECH>BRANCH5>SQUIRREL
```

specifies a file called SQUIRREL, in the subdirectory BRANCH5, in the top-level directory BEECH, on the disk FOREST. Figure 2-3 illustrates how the pathname leads through a tree of directories and files. (Segment directories and access categories do not appear in this diagram, although they could be parts of any tree.)

In specifying a pathname, do not include any spaces immediately before or after an angle bracket. For example,

```
Correct:   <HOUSE>DOOR>KNOB
Incorrect: <HOUSE >DOOR >  KNOB
```

The maximum length of a pathname allowed by PRIMOS is 128 characters.

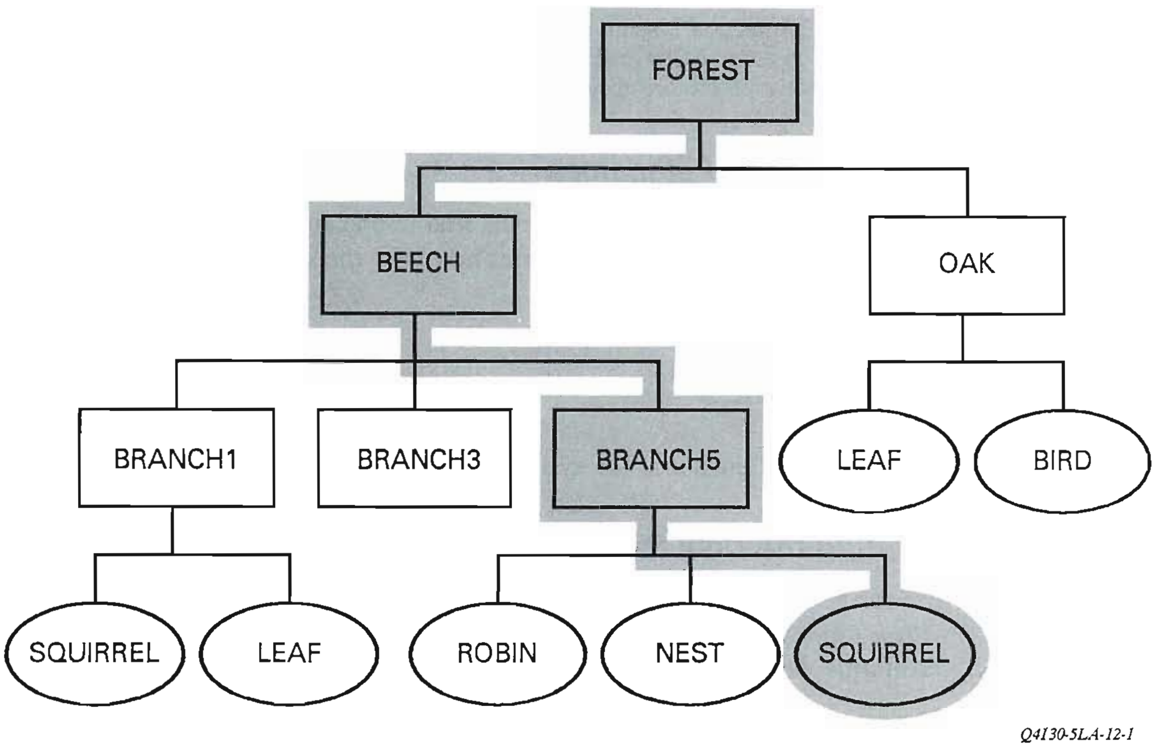


FIGURE 2-3
The Pathname <FOREST>BEECH>BRANCH5>SQUIRREL

MFD Pathnames: MFDs are top-level directories on each disk that list the other top-level directories. An MFD's name is always simply MFD. Therefore, the pathname of an MFD takes the form <*diskname*>MFD. For example, the pathname of the MFD on the disk HOUSE is <HOUSE>MFD. <HOUSE> alone is not a legal pathname, because a pathname must lead to a directory or other file system object. <HOUSE> specifies the disk name but does not specify any file system object.

Password-protected Directories: Some directories may have a password associated with them. When you use a passworded directory in a pathname, you must give the directory name followed by one blank space and the password:

directoryname password

You must also enclose the entire pathname in single quotation marks. For example, if the password *KEY* is associated with the directory DOOR, specify a pathname including the directory DOOR in the following form:

```
'<HOUSE>DOOR KEY>KNOB'
```

Spaces are permitted in a pathname only between a directory name and its password.

Note

The ACL protection system, explained in Chapter 5, is normally used instead of the directory password system. ACLs make pathnames with passwords and quotation marks unnecessary. More information on directory passwords appears in Appendix F.

Uniqueness of Pathnames

Each pathname must be unique; it must lead to one and only one file. This uniqueness means that two objects in different directories may have the same objectname, but that objects in the same directory may not.

Shortening Pathnames

PRIMOS can often find a file system object even if you don't specify an absolute pathname.

Omitting the Disk Name

You can usually specify a pathname without the disk name. In this case the pathname begins with a top-level directory name. Such a pathname is called a **full pathname** (or **ordinary pathname**, because it is the type of pathname most frequently used).

When you omit the disk name, PRIMOS searches logical disks one after another until it encounters a top-level directory with the same name as the first element in the pathname. As long as each top-level directory name is unique throughout all the logical disks, PRIMOS can always locate the correct top-level directory as it searches the disks.

Disks are searched in the order of their **disk numbers.** Each disk has a unique disk number. The disk number associated with each disk name can can be found with the STATUS DISKS command, described in Chapter 19. The search begins with the lowest numbered disk and continues to higher numbered disks until the target top-level directory name is found.

Suppose, for example, that a system has logical disks numbered 0 through 61 and that there is only one top-level directory named BEECH, located on the disk FOREST, which has disk number 3. You can then give the pathname

```
<FOREST>BEECH>BRANCH>SQUIRREL
```

as

```
BEECH>BRANCH>SQUIRREL
```

In this case, PRIMOS first searches disks 0, 1, and 2 and then finds the top-level directory BEECH on disk 3. As long as only one top-level directory is named BEECH, the pathname is unambiguous.

If the top-level directory name BEECH occurs on more than one disk, PRIMOS may not find the correct directory. PRIMOS finds the top-level directory BEECH with the lowest disk number. Although PRIMOS permits a top-level directory name to appear on more than one disk, System Administrators usually encourage unique top-level names, so that ordinary pathnames always lead to the right directory.

The Current and Origin Directories

When you work with PRIMOS, you are always **attached** to a specific directory in the file system hierarchy. The directory to which you are attached at any moment is called your **current directory**.

Relative Pathnames

Your current directory is a point of reference for any pathnames you give PRIMOS. You can often specify pathnames relative to the current directory, rather than relative to a top-level directory. Such pathnames are called **relative pathnames.** Relative pathnames begin with the symbol *> which stands for everything in the pathname down to and including the current directory name. For example, when the current directory is BEECH>BRANCH5, the pathnames

```
BEECH>BRANCH5>TWIG9>LEAF3
```

and

```
*>TWIG9>LEAF3
```

have the same meaning.

The directory you are attached to when you log in is called your **origin directory** or **Initial Attach Point** (IAP). Your origin directory is a top-level directory or subdirectory that contains files and directories you frequently use. The System Administrator assigns you an origin directory.

PRIMOS provides commands, documented in the next chapter, that allow you to attach to other directories after you log in.

Current Disk

When top-level directory names are not unique, you may need to give absolute pathnames. You can shorten absolute pathnames that begin with the name of your current disk by using the current disk symbol <*>. Your current disk is the one that contains the directory you are currently attached to. You can substitute <*> for the name of your current disk at the beginning of an absolute pathname. For example, if the current disk is <FOREST> then

```
<*>BEECH>BRANCH5
```

and

```
<FOREST>BEECH>BRANCH5
```

are equivalent.

Note

Do not confuse <*>, meaning current disk, with *>, which means everything in the pathname down to and including the current directory.

Pathnames, Objectnames, and PRIMOS Commands

You use objectnames and pathnames frequently as arguments to PRIMOS commands. With commands that take pathname arguments, you can give either absolute, ordinary, or relative pathnames. Often you can give an objectname instead of a pathname.

- You must give a pathname to refer to objects not in the current directory.
- You must give a pathname when the command attaches you to another directory.
- You can give an objectname instead of a pathname to refer to objects in the current directory unless the command attaches you to another directory.

A few commands explicitly require the use of an objectname alone. These requirements are documented in the command descriptions given in this book.

3

PRIMOS File System Commands

This chapter introduces a set of commands that you can use to carry out basic operations on the PRIMOS file system. Because so much work with PRIMOS involves the file system, you frequently use these commands as you work with other PRIMOS features and subsystems. The basic file system commands allow you to

- List the contents of a directory (LD)
- Attach to another directory (ATTACH)
- Return to the origin directory (ORIGIN)
- Create new directories (CREATE)
- Examine files (SLIST)
- Rename file system objects (CNAME)
- Copy file system objects (COPY)
- Delete unwanted file system objects (DELETE)
- Protect file system objects from accidental deletion (SET_DELETE)
- Record everything you do during terminal sessions (COMOUTPUT)
- Clean up after Ctrl P (RELEASE_LEVEL, CLOSE –ALL)

Note

If you receive the error message `Insufficient access rights` when you try to execute one of these commands, see your System Administrator. The command descriptions in this chapter list the access rights specifically required by each command. Chapter 5 fully explains access rights.

Examining Directory Contents

You can examine the contents of a directory with the LD command. The simplest format is

```
LD
```

In this format, the LD command displays a list of all file system objects in your current directory. Objects are listed by type in the following order: files, segment directories, directories, and access categories. Objects within each category are sorted alphabetically.

Suppose, for example, that your current directory is called SMITH. When you give the LD command, you see a display of the contents of SMITH:

```
OK, LD

<DISKA>SMITH (LUR access)
32 records in this directory, 163 total records out of quota of 500.

7 Files.

ABBREV         BUDGET1          BUDGET2          MAIL
LETTER         PAYROLL.INFO     REPORT

3 Segment Directories.

EXAMPLE.SEG       PROGRAM.SEG      TABULATION.SEG

2 Directories.

SUB1              SUB2

2 Access Categories.

GENERAL.ACAT       LETTER.ACAT
```

The first two lines of the display give additional information about the directory:

- The first item in the display is the absolute pathname of the directory. In this case the pathname is `<DISKA>SMITH`.
- The second item shows your access rights to the directory. In this case you have LUR access. Access rights are explained in Chapter 5. If the directory is not ACL protected, `(Owner)` or `(Non-owner)` appears. These terms are explained in Appendix F.
- Items on the second line give information about the size of the directory. `records` shows the number of records in the directory itself (in this case 32). `total records` shows the total number of records in the directory and all the subdirectories below it in the tree (in this case 163). `quota` shows the maximum total records permitted (in this case 500). If the quota is 0, there is no maximum limit. These terms are discussed further in Chapter 8.

Arguments and Options for the LD Command

The LD command can take objectnames and pathnames for arguments, allowing you to list the names of specific objects and the contents of directories other than the current directory. Such arguments usually incorporate **wildcards**, characters that allow an argument to refer to a group of objects rather than a single object. Chapter 6 explains the use of wildcard arguments with the LD command.

The LD command also allows several options that request additional information about directory entries, such as user access rights, date and time created, and size. For example, you can use the –PROTECT option to find out whether files have been protected from accidental deletion. The –PROTECT option is discussed below in the section Protecting Files From Accidental Deletion. The *PRIMOS Commands Reference Guide* gives complete information about options to the LD command.

Access Requirements

You must have List (L) access to a directory in order to list its contents. If you attempt to use the LD command on a directory to which you do not have List access, you receive the message

```
No information. (current-directory) (ld)
```

Attaching to Another Directory

Use the ATTACH command to move from your current directory to another directory in the file system. When you attach to a new directory, it becomes your current directory. The format of the ATTACH command is

ATTACH *new-directory*

where *new-directory* is the pathname of the directory that becomes your new current directory.

For example, if you want to attach to a directory with pathname <FOREST>BEECH, give the command

```
OK, ATTACH <FOREST>BEECH
```

The ATTACH command always requires either a full, ordinary, or relative pathname argument. Even when the target directory is within your current directory, you must give at least a relative pathname. You cannot give the target directory's objectname alone.

For example, if the current directory is the top-level directory CITY, and the target directory has the pathname CITY>CHICAGO, you must give the ATTACH command as

```
OK, ATTACH *>CHICAGO
```

or

```
OK, ATTACH CITY>CHICAGO
```

You cannot give the command as

```
OK, ATTACH CHICAGO
```

If you do give the directory name alone, PRIMOS treats it as a top-level directory name. In the last example above, PRIMOS attempts to attach you to a top-level directory called CHICAGO instead of to CITY>CHICAGO. PRIMOS does this because it expects a pathname argument, and a single objectname can only be a pathname for a top-level directory.

Note that ATTACH is different from most other commands in this respect. With other PRIMOS commands you can often give an objectname alone, instead of a relative pathname, when the target object is within the current directory.

Attaching Errors

If PRIMOS cannot attach you to the directory you request, you remain attached to your current directory.

If an attempt to ATTACH to a subdirectory fails because the directory is not found, PRIMOS returns the following error message:

```
Not found.   directory-name (ATTACH)
```

This error often occurs because you have mispelled the directory name or some other part of the pathname.

An attempt to ATTACH to a top-level directory can fail because the directory does not exist, because PRIMOS cannot reach the system where it does exist, or because you do not have the right to attach to it. In any of these cases, PRIMOS returns the following error message:

```
Top-level directory not found or inaccessible. directory-name (ATTACH)
```

If the ATTACH command fails because you don't have sufficient access rights to a subdirectory, PRIMOS returns the following error message:

```
Insufficient access rights.  directory-name (ATTACH)
```

If you give an incorrect password with the ATTACH command for a directory that requires passwords, PRIMOS returns the following error message:

```
Bad Password.  directory-name (df_unit_)
```

Access Requirements

To attach to an ACL-protected directory, you must have Use (U) access to it.

To attach to a passworded directory, you must supply any necessary passwords in the pathname, as in

```
OK, ATTACH 'BEECH SECRET>BRANCH5'
```

Pathnames with passwords are described in Chapter 2. For more information on passworded directories, consult Appendix F.

Returning to Your Origin Directory

To return to your origin directory, use the ORIGIN command. The format is

ORIGIN

For example, if your origin directory is the top-level directory QUARTET on the disk <MUSIC>, then the command

```
OK, ORIGIN
```

is equivalent to

```
OK, ATTACH <MUSIC>QUARTET
```

Creating New Directories

Use the CREATE command to create new subdirectories within a directory. By creating new subdirectories, you can organize your files in the tree structure provided by the PRIMOS file system. The format of the CREATE command is

CREATE *pathname*

where *pathname* may be

- The objectname of a new subdirectory to be created within the current directory
- The pathname of a new subdirectory to be created within some other directory

For example, if your current directory is BEECH, then

```
OK, CREATE BRANCH6
```

creates the subdirectory BRANCH6 in the directory BEECH.

Note the important difference between CREATE and ATTACH.

- ATTACH treats an argument that contains an objectname alone as a top-level directory name.
- CREATE treats such an objectname as the name of a subdirectory to be created within the current directory.

To create a new subdirectory within a directory other than the current directory, you must specify either a full, ordinary, or relative pathname. For example, if ELM is a subdirectory of the top-level directory TREES, then

```
OK, CREATE TREES>ELM>BRANCH1
```

creates the subdirectory BRANCH1 in the directory ELM.

You can use a relative pathname to create a new directory below your current directory in the file system tree. For example, suppose your current directory is BEECH. If BEECH contains the subdirectory BRANCH4, you can create a subdirectory called LEAF within BRANCH4 using

```
OK, CREATE *>BRANCH4>LEAF
```

Two objects of the same name are not permitted in a directory. If you try to create a subdirectory with the same name as some other object in the same directory, PRIMOS returns the following message:

```
Already exists.  directory-name  (CREATE)
ER!
```

Access Requirements

You must have Add (A) access rights to the directory in which you create a new subdirectory.

Note

You can create files in your directories in a variety of ways. You can create and modify **text files**, including documents, programs, and data lists, using one of the text editors supported by PRIMOS: EDITOR and EMACS. Text editors are discussed in Chapter 4. You can create a file that records your terminal input and output with the COMOUTPUT command discussed below. Many of the programs and subsystems discussed later in this book also create files of various types.

Examining the Contents of a File

You can examine the contents of any text file at the terminal with the SLIST command. The format is

SLIST ***pathname***

The file specified by *pathname* is displayed at the terminal. SLIST treats an objectname alone as a file within the current directory. For example,

```
OK, SLIST LETTER
```

and

```
OK, SLIST *>LETTER
```

both display the contents of a file called LETTER (if it exists) in the current directory.

SLIST displays the file by scrolling it continuously on your terminal screen. You can stop and restart the terminal display using Ctrl S and Ctrl Q, as discussed in Chapter 1, as long as you have previously used the TERM –XOFF command.

Access Requirements

To display the contents of a file using SLIST you must have Read (R) access to the file.

Renaming File System Objects

Use the CNAME command to change the name of a file system object. This format is

CNAME ***old-name new-name***

old-name is the pathname of the object to be renamed, and *new-name* is the new objectname for the object. *new-name* must be an objectname; it cannot be a pathname. For example,

```
OK, CNAME REPORTS>DRAFT FIRSTDRAFT
```

changes the name of the entry called DRAFT in the top-level directory REPORTS to FIRSTDRAFT.

If *new-name* already exists, PRIMOS displays the following message:

```
Already exists.  objectname  (CNAME)
ER!
```

A misspelled *old-name*, such as DRIFT instead of DRAFT, produces the `Not found` message:

```
OK, CNAME REPORTS>DRIFT FIRSTDRAFT
Not found.  DRIFT  (CNAME)
ER!
```

When changing the name of an object in the current directory, you can substitute the objectname alone for the pathname argument. For example, if you are already attached to the directory REPORTS, then the commands

```
OK, CNAME *>DRAFT FIRSTDRAFT
```

and

```
OK, CN DRAFT FIRSTDRAFT
```

are equivalent.

Access Requirements

To change the name of a file system object, you must have Delete (D) and Add (A) access to the directory that contains it.

Copying File System Objects

Use the COPY command to copy file system objects, either within the same directory or from one directory to another. In its simplest form, the format is

COPY ***pathname*** **[*new-pathname*]**

Argument	*Description*
pathname	Indicates the name of the object to be copied. The object itself is not removed from its directory or altered in any way. If the object to be copied is in the current directory, then you can give the objectname alone instead of the pathname.
new-pathname	Indicates the pathname of the new copy. If you are copying the object into the current directory, then you can give the new objectname alone instead of the pathname. If *new-pathname* is not specified, the object is copied into the current directory under its original name. Note, however, that you must give a new name when copying an object within the same directory, because names must be unique within a directory.

COPY Examples

In the first example, LETTER is a file in the current directory MAIL. The command

```
OK, COPY LETTER LETTER.NEW
```

creates a second copy of LETTER in the directory MAIL under the name LETTER.NEW. The original file LETTER remains in the directory. Figures 3-1 and 3-2 illustrate the directory MAIL before and after the COPY command is given.

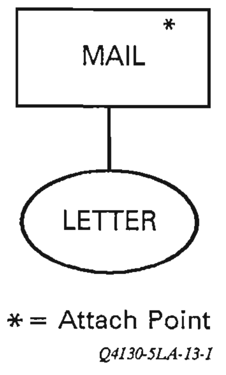


FIGURE 3-1
Directory Before COPY Command Is Given

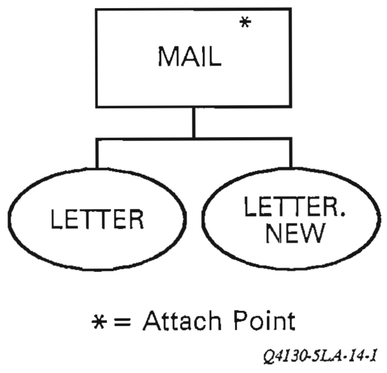


FIGURE 3-2
Directory After COPY Command Is Given

A second example illustrates copying from one directory to another. To copy BIRDSNEST, a subdirectory of BEECH, to the directory TREEHOUSE, use the command

```
OK, COPY BEECH>BIRDSNEST TREEHOUSE>BIRDSNEST
```

If you are currently attached to TREEHOUSE, then you can shorten the command to

```
OK, COPY BEECH>BIRDSNEST
```

If you are currently attached to BEECH, the same operation can be accomplished with

```
OK, COPY BIRDSNEST TREEHOUSE>BIRDSNEST
```

Figures 3-3 and 3-4 illustrate the directory BEECH before and after the copy operation defined by these commands.

Note that copying a directory copies all subdirectories and files within the directory as well. When you copy a directory, PRIMOS first queries you for permission. For example,

```
OK to copy directory "BEECH>BIRDSNEST" to "BIRDSNEST"?
```

Note

If you try to copy a file over another file that already exists, PRIMOS displays the message

```
"pathname" already exists, do you wish to overwrite it?
```

If your directory or directory tree does not have enough storage room to hold the object you wish to copy, PRIMOS displays the message `Maximum quota exceeded` and does not execute the copy operation. You need to delete some objects to make room. See the next section, Deleting File System Objects. See Chapter 8 for further explanation of quotas.

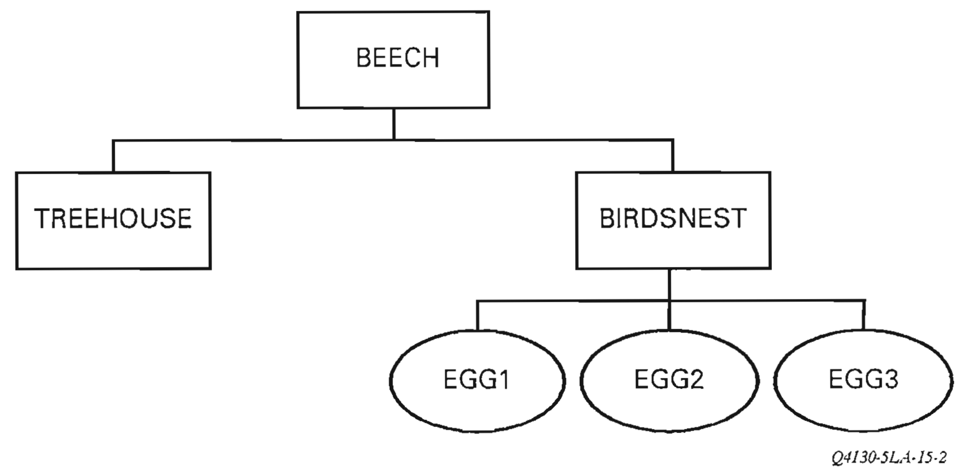


FIGURE 3-3
Directory Before COPY Command Is Given

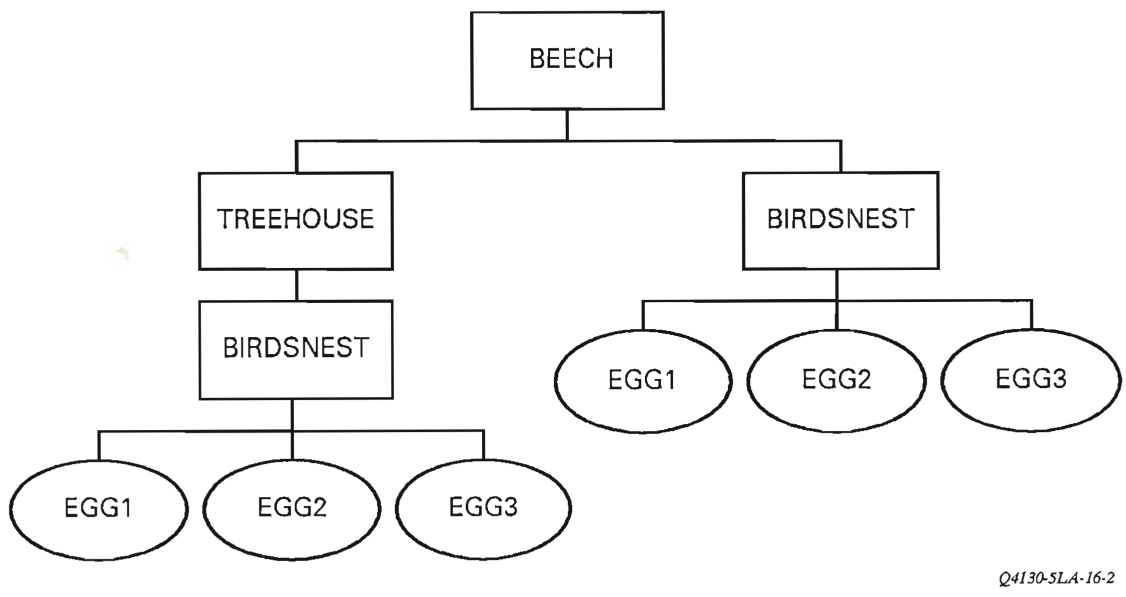


FIGURE 3-4
Directory After COPY Command Is Given

Options for the COPY Command

The COPY command allows a number of options that affect the copying process. These options are discussed in the *PRIMOS Commands Reference Guide*.

Access Requirements

To copy a file, you must have Read (R) access rights to the file. You must also have Add (A) access rights to the directory containing *new-pathname*. If an object with the new name already exists in the target directory, you must also have Delete (D) access to the target directory, so that the new entry can overwrite the old one.

Deleting File System Objects

When you no longer need file system objects, you can remove them from a directory with the DELETE command to provide more room for other work. The format is

DELETE *pathname*

If the object to be deleted is within the current directory, then you can substitute the objectname alone for *pathname*. If the object is a directory or an access category, PRIMOS queries you to double-check that the command is intended for this entry. For example, if SUBDIR5 is a subdirectory of the current directory then the command

```
OK, DELETE SUBDIR5
```

results in the following query from PRIMOS:

```
OK to delete directory "SUBDIR5"?
```

If you want the directory deleted, you must reply with Y or YES. If you reply with N or NO, the object is not deleted.

If the object specified is not found, PRIMOS returns a `Not found` message. For example,

```
OK, DELETE BRANCH23
Not found.  "BRANCH23" (delete)
```

You cannot delete the directory to which you are attached or one of its parent directories. If you attempt this, you receive a `File open on delete` or `Directory not empty` error message.

Options for the DELETE Command

The DELETE command has several options that tell PRIMOS how to proceed with deletion activities. These options are discussed in *PRIMOS Commands Reference Guide*.

Access

You must have Delete (D) access to the directory containing the object you want to delete.

Note

In order to delete a segment directory or an EPF you also need Write (W) access to the object.

Protecting Objects From Accidental Deletion

To prevent accidental deletion of a file, segment directory, or directory, use the SET_DELETE command. PRIMOS queries you before deleting any object that has been delete-protected with SET_DELETE. SET_DELETE works only for objects in directories protected by ACLs.

The format is

$$\text{SET\_DELETE } \textit{pathname} \left[\begin{Bmatrix} \text{-PROTECT} \\ \text{-NO\_PROTECT} \end{Bmatrix} \right]$$

Argument/Option	*Description*
pathname	Names the object to be delete-protected. For an object in the current directory, the objectname alone may be substituted.
-PROTECT	Causes PRIMOS to query you when you attempt to delete the object.
-NO_PROTECT	Removes delete-protection provided by SET_DELETE. This is the default state in which file system objects are created.

If you give the SET_DELETE command without either option, -PROTECT is assumed and the object specified is delete-protected.

In the example below, the file IMPORTANT in the current directory is delete-protected. The DELETE command queries you before deleting the object:

```
OK, SET_DELETE IMPORTANT -PROTECT
OK, DELETE IMPORTANT
"IMPORTANT" protected, ok to force delete?  NO
File is delete-protected. Unable to delete "IMPORTANT" (delete)
OK,
```

The same querying occurs if you try to delete a directory that contains protected objects (even though the directory itself is not delete-protected). For example, assume that your current directory contains the subdirectory PAPERS and that PAPERS in turn contains the file IMPORTANT.

```
OK, SET_DELETE *>PAPERS>IMPORTANT
OK, DELETE PAPERS
Ok to delete directory "PAPERS"? YES
"*>PAPERS>IMPORTANT" protected, Ok to force delete? YES
OK,
```

If you answer NO to the last query, you receive the following response:

```
The directory is not empty.  Unable to delete "PAPERS" (delete)
```

In effect, a parent directory is delete-protected if any of the objects it contains is delete-protected. The converse of this is not true. Objects within delete-protected directories are not themselves delete-protected unless they have been protected explicitly by SET_DELETE.

For example, suppose that the directory BASKET, containing the files EGG1, EGG2, EGG3, and EGG4, has been delete-protected with the following command:

```
OK, SET_DELETE BASKET
```

If you attempt to delete BASKET and the files it contains using

```
OK, DELETE BASKET
```

you receive a delete-protect query from PRIMOS. However, you can still delete all of the EGG files individually with

```
OK, DELETE EGG1;DELETE EGG2;
```

and so on, without receiving any warning from PRIMOS.

Determining Whether Objects Are Delete-protected

To see whether an object is delete-protected, use the –PROTECT option of the LD command. The format is

LD –PROTECT

If the object is delete-protected, the letters *dprot* appear on the information line for the object. The following example shows the delete-protect status of objects in the current directory TREES:

```
OK, LD -PROTECT

<FOREST>TREES (ALL access)
63 records in this directory, 5779 total records out of quota of 0.

6 Files.
name                   access delprot   type  rbf    protected

--------------------------------------------------------------------
ACACIA                  ALL                 sam        (Default ACL)
BEECH                   ALL                 sam        (Default ACL)
CHESTNUT                ALL                 sam        (Default ACL)
HEMLOCK                 ALL                 sam        (Default ACL)
MAPLE                   ALL                 sam        (Default ACL)
OAK                     ALL   dprot         sam
OK,
```

In this example, the file OAK is delete-protected.

Access Requirements

You need Delete (D) access to the directory that contains any object you want to delete-protect with the SET_DELETE command.

Recording Terminal Sessions

You can make a record of an interactive terminal session with the COMOUTPUT command. This can be useful when you have problems and want to record the screen dialog to analyze it or show it to another person. You can also use COMOUTPUT to save the screen output from a PRIMOS command such as LD. To start recording a terminal session use the command

```
COMOUTPUT pathname
```

This creates a file named *pathname* and starts recording in it whatever you type at the terminal and whatever responses the system makes to your commands. The commands and responses continue to appear on your terminal as well.

The file specified by *pathname* is called a **command output** or **COMO file**. The standard filename suffix for COMO files is .COMO, but you can specify a pathname without this suffix if you wish.

You can create a COMO file in the current directory by specifying the filename alone instead of the pathname.

To stop recording material and close the file, give the command

```
COMOUTPUT -END
```

Once you have closed the file, you can read it with the SLIST command, edit it with a text editor, or request a printed copy using the SPOOL command (discussed in Chapter 4).

The following example illustrates the creation of a command output file:

```
OK, COMOUTPUT SESSION.COMO
OK, LD

<DISKA>BEETHOVEN>QUARTETS (DALURWX access)
54 records in this directory, 54 total records out of quota of 0.

3 Files.

EARLY                LATE                 MIDDLE

OK, COMO -END
OK,
```

A file named SESSION.COMO now exists in the current directory. The file contains the record of all terminal input and PRIMOS output following the COMOUTPUT

SESSION.COMO command, including the COMO –END command. Listing SESSION.COMO with the SLIST command reveals the contents as follows:

```
OK, SLIST SESSION.COMO
OK, LD

<DISKA>BEETHOVEN>QUARTETS (DALURWX access)
54 records in this directory, 54 total records out of quota of 0.

3 Files.

EARLY            LATE             MIDDLE

OK, COMO -END
OK,
```

The final command, COMO –END, is part of the recorded material, but the initial command, COMOUTPUT SESSION.COMO, is not.

Note

If a file named *pathname* already exists, it is normally overwritten by the COMOUTPUT command. However, you can specify that the new material be placed at the end of an existing file. For details on this and other extensions of the COMOUTPUT command, see Chapter 14.

Access Requirements

You must have Add (A) access to the directory in which you wish to create a new COMO file. You must have Write (W) access if you want to overwrite an existing file.

Interrupting Commands

Chapter 1 notes that you can interrupt a command in progress with Ctrl P. For example,

```
OK, LD

<DISK>JUDY (ALL)
91 records in this directory, 135 total records out of quota of 500.

39 Files.

FILE1            FILE2            FILE3            FILE4
FILE5            FILE6            FILE7            FILE8
FILE9

Ctrl P

QUIT.
OK,
```

Interrupting programs and commands with Ctrl P can leave files open and may lead to unwanted results in the behavior of subsequent commands. Use the following commands to clean up after interrupting a command with Ctrl P:

```
CLOSE -ALL
```

and

```
RELEASE_LEVEL
```

The first command closes any files the interrupted command may have left open. The second frees resources for future use. See Chapter 12 for more information on the use of RELEASE_LEVEL and CLOSE –ALL.

Note

You may be able to restart an interrupted command with the START command. If you want to restart a command, don't use the CLOSE –ALL or RELEASE_LEVEL commands. See Chapter 12 for information on the START command.

Creating and Printing Files

Text files are the most widely used PRIMOS files. This chapter shows you how to create and modify text files with the EDITOR text editor, and how to print them with the SPOOL command. The chapter also briefly introduces the EMACS text editor.

Text Files

Text files are files that contain **character** data. Character data includes letters, numbers, and other symbols as well as control characters.

Note

Control characters are special characters that often cannot be displayed on a terminal screen or printed on a printer. Many control characters can be generated by pressing $\boxed{\text{Ctrl}}$ in combination with another key at the keyboard. Control characters are often used to control display devices and printers.

Some examples of text files are

- Documents, manuscripts, and letters
- Program source code
- Data lists and tables

PRIMOS and subsystems often use text files to do their work. For example, a **command input file** is simply a text file containing commands that PRIMOS can process without input from the terminal. (Chapter 14 discusses command input files.) Compilers use text files containing program source code to create binary files that can be linked, loaded, and executed by the computer.

Text Editors

You can use a text editor to create and modify text files. Prime offers PRIMOS users two text editors.

- EDITOR is a **line-oriented** text editor. You use it to create or modify text files one line at a time. Because EDITOR is line-oriented, you use it in exactly the same way on all kinds of terminals, including line-oriented terminals such as teletypes. The next section describes the major features of EDITOR and shows you how to use them.
- EMACS, a separately priced product, is a **screen oriented** text editor. You use it, like a word processor, to create and modify text by manipulating the text on the terminal screen. EMACS features include
  - Automated search for and replacement of selected text
  - Simultaneous editing of several files, allowing transfer of text between files
  - User defined macros that allow you to link frequently used key sequences to a single key
  - Special editing modes to simplify the editing of text that must follow specific formats, such as program source code

  EMACS is not extensively documented in this book. See the *EMACS Primer* and the *EMACS Reference Guide* for complete information.

The Editing Process

Editing and Saving: When you use an editor to create a new file, you first create the file in memory and then save it on disk. When you modify an old file, the editor first copies the old file from disk to memory where it can be edited. The original version remains intact on the disk until the text editor copies the new version back onto the disk. In fact, both EMACS and ED allow you to save the new version of a file without overwriting the old version. Figure 4-1 illustrates the editing process.

Because editing takes place in memory, you can freely modify text without fear of destroying or losing the original text. However, this process also means that you must be sure to save text that you want to keep. If you attempt to quit EMACS or ED without saving new or modified text, both programs warn you and ask if you want to save the new material before actually quitting.

The Editing Window: When you are editing a file, both editors provide you with a **window** on the file, a section of the file that appears on the screen. In the case of ED, the window shows you a single line of the file at a time. With EMACS you see several lines at once, enough to fill most of your terminal screen. Both editors allow you to move the window backwards and forwards through the file to view different sections. Editing operations can be carried out on the portion of the file visible in the editing window. Figure 4-2 shows how the editing window displays a portion of the file on the screen.

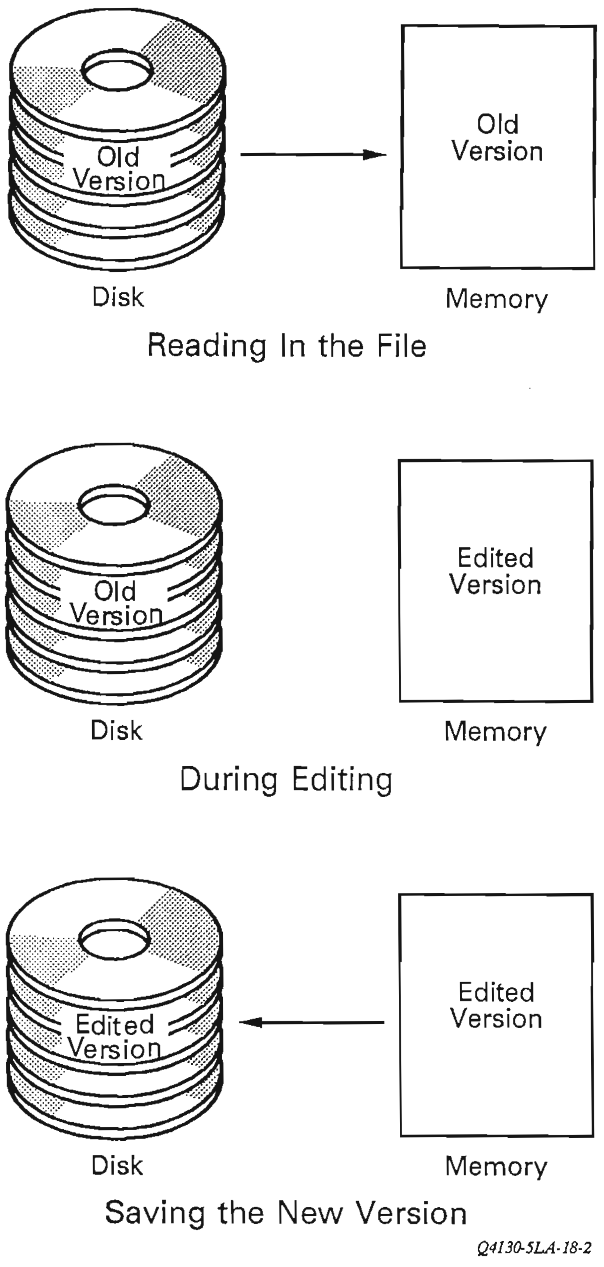


FIGURE 4-1
The Editing Process

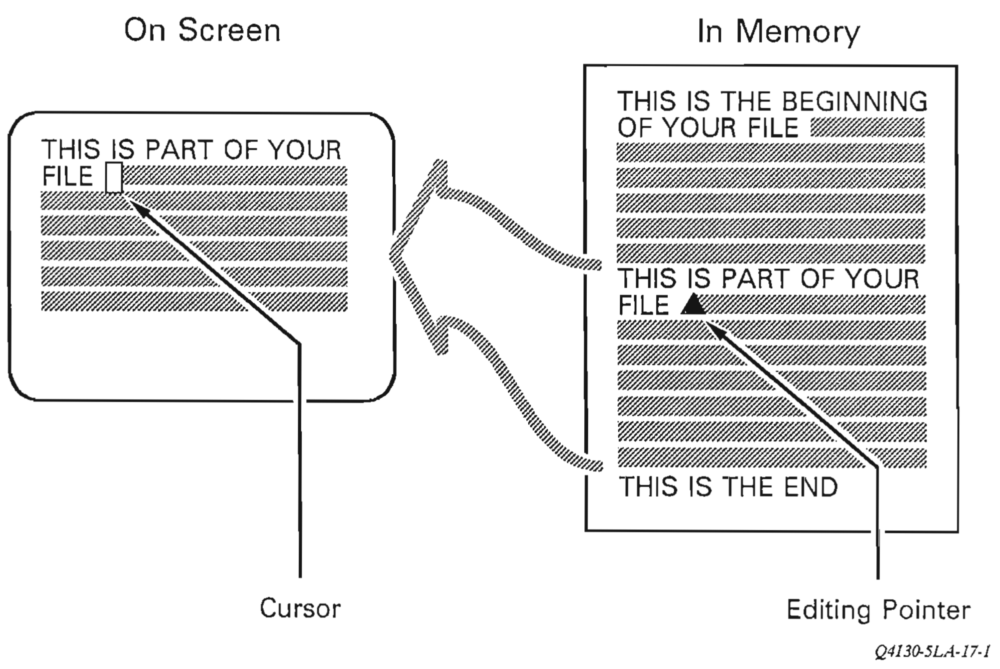

On Screen
In Memory
THIS IS PART OF YOUR FILE
THIS IS THE BEGINNING OF YOUR FILE
THIS IS PART OF YOUR FILE
THIS IS THE END
Cursor
Editing Pointer
Q4130-5LA-17-1

FIGURE 4-2
The Editing Window

Document Processing

You can use both ED and EMACS with a **text formatter**, such as **RUNOFF**, to create printed documents in a wide variety of formats. To create a document, you first use a text editor to create a file containing the text and formatting commands. The **formatting commands** specify such characteristics as page layout, page headings, type faces, and document sectioning. The text formatter then processes this file and automatically produces a second file that contains your text along with printer control information. When this file is sent to the printer, the printer uses the printer control information inserted by the text formatter to print a document with the characteristics you have specified.

For example, if you want to underline a section of text in a document to be processed by the RUNOFF formatter, you insert double braces ({{ }}) around the text. To underline the word *underline*, for example, you type `{{underline}}` in the original text file. RUNOFF interprets double braces as the command to underline the text. When RUNOFF processes your file, it inserts printer control codes or other information that cause the printer to print the word as underline. Information about the RUNOFF text formatter is available in the *New User's Guide to EDITOR and RUNOFF*.

Note

If you have used a word processor, the use of a text editor to prepare documents for printing is familiar to you. However, ED and EMACS alone are not intended for elaborate document formatting. For example, they do not provide a way to select type fonts and styles, although EMACS can be used to format text between margins and in columns on the screen much like a word processor. For extensive document formatting, you need to use the editors with a text formatter as explained above.

Using EDITOR

You edit a text file with EDITOR (ED) by entering text and giving commands at the terminal. Because ED is line-oriented, you don't manipulate text on the screen. Instead, you use ED commands to carry out operations such as deleting lines, searching for strings, and so on. You can carry out most editing tasks quite efficiently using about 20 commands. This section explains the most important ED commands. For a complete reference, see the *New User's Guide to EDITOR and RUNOFF.*

Invoking ED

To begin editing a file, you invoke ED with the ED command. To create a new file the format is

ED

To modify an existing file the command is

ED ***pathname***

where *pathname* is the name of an existing file. If the file is in your current directory, you can give the filename alone.

Note

The term **invoke**, as used throughout this book, means to initiate the operation of a program or subsystem. You invoke subsystems and programs by giving a PRIMOS command (in this case the ED command). In the case of an interactive subsystem like ED, invoking ED takes you out of PRIMOS command level and into the subsystem. While you are working in a subsystem, you can give only the commands recognized by the subsystem. (Commands recognized by a subsystem are often called subcommands to distinguish them from PRIMOS commands.) This chapter introduces the basic commands recognized by ED.

ED Modes

When you are working with ED, you give ED two different kinds of input. Sometimes what you type at the terminal is text to be added to the file being edited; sometimes it is an ED command, instructing ED to carry out an editing operation such as deleting or moving a line of text. In order to distinguish between these types of input, ED operates in two **modes** called INPUT mode and EDIT mode.

- In **INPUT mode**, user input is treated as text and added to the file being edited. When ED first enters INPUT mode, the word `INPUT` is displayed at your terminal. When you begin to edit a new file by invoking ED without a filename, ED starts in INPUT mode.
- In **EDIT mode**, user input is interpreted as an ED command, and ED responds by carrying out the editing operation specified. When ED first enters EDIT mode, the word `EDIT` appears on your screen. When you start to edit an old file by invoking ED with a filename, ED begins in EDIT mode.

Switching Modes: You can switch modes at any time by typing a Return by itself, not preceeded by any other characters. You can also use the semicolon character (;) as explained in the section Special Characters, below. In a typical file editing session, you enter some text in INPUT mode, switch to EDIT mode to make changes and corrections, return to INPUT mode to enter more text, and so on. You end the session in EDIT mode, giving ED a command to save your work in a disk file.

Entering ED Commands and Text

You enter commands and text one line at a time, ending each line with a Return. A line that contains only a Return is called an **empty line**. Empty lines cause ED to switch modes as described above.

Special Characters: ED allows you to use the PRIMOS erase and kill characters to edit a line of input before you press Return. These characters work exactly as they do at PRIMOS command level. If you have changed the PRIMOS erase and kill characters using the PRIMOS TERM command, then the characters you have selected work under ED as well. See Chapter 1 for an explanation of the erase and kill characters.

Note

A line that ends with a kill character is treated as an empty line and causes ED to switch modes.

ED recognizes two other special characters in the input line.

Character	*Description*
;	ED treats the semicolon as if it were a Return. When ED encounters a semicolon in an input line, it begins a new line. A semicolon without any preceding characters, or preceded by another semicolon, generates an empty line and causes ED to switch modes. You can use a semicolon in EDIT mode to put multiple ED commands on a single line.
^	The caret is the ED escape character. The **escape character** allows you to include characters such as double quotation marks, question marks, and semicolons, which have special functions, in your input text. When you include an escape character in an input line, ED treats the following character as part of the text, ignoring any special function it normally has. The escape character itself does not become part of the text. For example, to include the line `"speak", he said` in your file, type `^"speak^", he said`

To include a caret in your text, type two carets in a row (^^).

Note

A character, such as double quote ("), used as an ordinary text element, rather than as a special character, is called a **literal character**. The escape character causes any following character to be treated as a literal character.

Entering Text in INPUT Mode

Use INPUT mode to create an entirely new file, to add more text to a file that you are currently editing, or to add text to a file that already exists on disk.

- If you are beginning a new file, enter INPUT mode immediately by invoking ED without a filename.
- If you are adding new text to a file that already exists, invoke ED with a filename. In this case ED begins in EDIT mode and you must switch to INPUT mode by typing an empty line.
- If you are giving ED commands in EDIT mode and you want to add more text to a file, you must switch to INPUT mode by typing an empty line.

When ED enters INPUT mode it prompts you with

```
INPUT
```

You can then begin to type your text.

A Sample File

The following FORTRAN program text illustrates the editing procedures described in this chapter:

```
C This program generates the numbers 1 to 10
C and prints the numbers on the terminal screen.
C
      DIMENSION LNUMB (50)
        DO 100 I = 1, 10
           LNUMB (I) = I
           WRITE (1, 200) LNUMB (I)
200   FORMAT (10X, I5)
100   CONTINUE
      CALL EXIT
      END
```

To enter this text for the first time, invoke ED without a filename by typing

```
OK, ED
```

ED then prompts you with

```
INPUT
```

Enter the text, ending each line with Return. Use the erase and kill characters to correct any errors that you notice before you press Return. Errors that you don't correct before typing Return must be corrected later in EDIT mode.

When you have entered the text, enter EDIT mode by typing an empty line. (That is, after typing the last line of the file, type Return to end the line and then *another* Return to switch to EDIT mode.) Once you are in EDIT mode, you can begin to edit the text immediately, save the text with the SAVE command, or save and quit with the FILE command. (SAVE and FILE are discussed in the section Ending and Saving, below.)

Working in EDIT Mode

ED provides over fifty commands that you can use to edit your file in EDIT mode, but you can probably do the majority of your work with the 22 commands introduced in this chapter. The editing process is an interactive dialog; you type an ED command, and ED responds either by performing the operation requested or by displaying an error message on the screen if there is some problem. Editing commands are **line-oriented**; they operate either on one line or a designated number of lines of your text. The mechanics of this process are relatively simple.

The Current Line: In EDIT mode, ED maintains an internal pointer at one line of your file called the current line. The **current line** is the first line that is processed by any command that changes text. After any editing operation, the current line is the last line that was processed. Several commands move the current line pointer. For example, TOP and BOTTOM move the pointer to the beginning and end of your file.

ED also numbers the lines of your file. These numbers are internal. They are not a part of the file itself, and they don't normally appear on the screen, but you can use them to move ED's internal pointer to a specific line.

Commands to move the pointer usually cause the new current line to be displayed on the screen. The exception to this rule is a special line called a null line.

Null Lines: While you are editing a file, ED sometimes creates lines called **null lines** that contain no text or characters. ED uses null lines to mark points where it can insert new text in your file. When you use the TOP command to move the pointer to the beginning of the file, for example, the current line is a null line before the first line of text, rather than the first line itself.

If you ask ED to print a null line, it puts the word `.NULL.` on the screen. Null lines are purely internal to ED. They do not become part of your file when text is saved.

Switching From EDIT to INPUT Mode: You can reenter INPUT mode from EDIT mode at any time by typing an empty line. When you enter INPUT mode from EDIT mode, the text you type is inserted beginning immediately after the current line.

ED Command Format

ED commands, like PRIMOS commands, may appear either alone or with arguments. The format is either

COMMAND

or

COMMAND *argument* [*...argument*]

ED commands take three types of arguments:

Argument	*Description*
String	Many ED commands refer to specific items of text. For example, the FIND command locates a specified section of text. With such commands, you give the text in question as an argument to the command. The command to find the word *cat*, for example, is `FIND cat` An argument consisting of a sequence of text characters is called a **string argument.** A string consists of any sequence of letters, numbers, and keyboard symbols, including blanks. For example the string *#define MAXOP 14* is a sequence of 16 characters including two numerals and two blank spaces. Commands that take string arguments have the following format: COMMAND *string*
Numerical	ED commands can take numerical arguments that indicate the line number to be affected, how many lines are to be affected, or how many times to carry out the operation specified by the command. For example, PRINT 4 tells ED to display four lines beginning with the current line. Commands that take numerical arguments have the following format: COMMAND *n* Some ED commands can take both a numerical and a string argument.
Pathname	Some ED commands, such as FILE, which saves edited text in a file, take the pathname of a file as an argument. If the file is in the current directory, the filename alone can be used. Commands that take pathame and filename arguments have the following format: COMMAND *pathname*

ED commands, like PRIMOS commands, often have both short and long forms. Wherever a short form exists, it is shown in red type.

Multiple Commands: You can include multiple commands in a single line by separating them with semicolons (;), just as you do with PRIMOS commands in a PRIMOS command line.

ED Error Messages

ED is an interactive program. You give commands and other input, and ED responds. However, ED's prompts are much more limited than those you see at PRIMOS command level. Each time it enters a new mode, ED puts the mode name on the screen (INPUT or EDIT). Otherwise, you do not see any prompts from ED unless you make a mistake while typing a command in EDIT mode. If you give ED a command that it cannot understand, you receive one of the following error messages:

Error Message	*Meaning*
`?`	Your input could not be interpreted as any of ED's commands. This error often results from attempting to enter text while you are still in EDIT mode.
`BAD command`	*command* was recognized, but there were problems with the argument. For example, the format was incorrect, or the specified file could not be found.

Basic ED Commands

Table 4-1 lists the ED commands you are most likely to use. The following sections describe these commands in in detail. See Appendix E for a complete listing of ED commands. Also see the *New User's Guide to EDITOR and RUNOFF* for further information.

The PRINT Command

The PRINT command displays specified lines of your file on the terminal screen. The format is

PRINT [*n*]

If *n* is positive, it specifies the number of lines you want displayed, beginning with the current line. The last line displayed becomes the new current line. If *n* is omitted or given a value of 1, -1, or 0, only the current line is displayed. If *n* is negative, ED moves the pointer back *n* lines from the current line, and then displays one line, which becomes the new current line. The space between PRINT and *n* is optional.

For example,

```
PRINT 5
.NULL.
C This program generates the numbers 1 to 10
C and prints the numbers on the terminal screen.
C
      DIMENSION LNUMB (50)
PRINT 2
      DIMENSION LNUMB (50)
        DO 100 I = 1, 10
PRINT -2
      DIMENSION LNUMB (50)
```

TABLE 4-1
Commonly Used ED Commands

Command	*Function*
PRINT	Display lines
TOP **NEXT** **POINT** **BOTTOM**	Move the pointer
LOCATE **FIND** **NFIND** **FIND(*n*)** **NFIND(*n*)**	Find a string
APPEND **CHANGE** **DELETE** **INSERT** **IB** **RETYPE** **OOPS**	Modify text
UNLOAD **DUNLOAD** **LOAD**	Copy text to or from a file
QUIT **FILE** **SAVE**	Save text and end text processing

Moving the Pointer

The commands described in this section move ED's internal pointer to a specific line.

The TOP Command: The TOP command moves the pointer to a null line at the top of the file, just before the first line of text. The format of the TOP command is

TOP

For example,

```
TOP
PRINT
.NULL.
PRINT 2
.NULL.
C This program generates the numbers 1 to 10
```

The BOTTOM Command: The BOTTOM command moves the pointer to a null line at the bottom of the file, just past the last line of text. The format of the BOTTOM command is

```
BOTTOM
```

For example,

```
BOTTOM
PRINT
.NULL.
PRINT -3
      CALL EXIT
PRINT 5
      CALL EXIT
      END
BOTTOM
```

The NEXT Command: The NEXT command moves the pointer a specified number of lines and displays the new current line. The format of the NEXT command is

NEXT [*n*]

where *n* is the number of lines that the pointer is to be moved.

Positive values of *n* move the pointer toward the end of the file; negative values move the pointer toward the beginning. If *n* is 0 or unspecified, the value of *n* is treated as 1 and the pointer is moved 1 line. If *n* is great enough to move the pointer beyond the top or bottom null line, the pointer stops at the null line, and either TOP or BOTTOM is displayed.

For example,

```
TOP
NEXT
C This program generates the numbers 1 to 10
NEXT 5
           LNUMB (I) = I
BOTTOM
NEXT
BOTTOM
```

The WHERE Command: ED generates line numbers in order to keep track of the location of the pointer in your file. The line numbers do not normally appear in your file when lines are displayed, and they do not become part of your file when it is saved on disk. You can find the current line number with the WHERE command. The format is

```
WHERE
```

You can have ED print line numbers when it prints lines by using the MODE NUMBER command. See the *New User's Guide to EDITOR and RUNOFF* for more information on MODE commands.

The POINT Command: The POINT command positions the pointer at a specified line and displays the line. The format of the POINT command is

POINT *n*

where *n* is the number of the line to which the pointer is to be moved.

The POINT *n* command is equivalent to the sequence TOP, NEXT *n*. The value of *n* must be greater than 0. If *n* is greater than the number of lines in the file, the pointer is left at the bottom.

For example,

```
POINT 5
        DO 100 I = 1, 10
POINT 7
           WRITE (1, 200) LNUMB (I)
POINT -4
BAD POINT
POINT 2
C and prints the numbers on the terminal screen.
```

Note

If you are using line numbers in your text (when writing a program, for example) they may not correspond to ED's line numbers. For example, the command POINT 200 does not point to the line

```
200    FORMAT  (10X, I5)
```

in the sample file. Instead, it positions the pointer at the bottom of the file, because the sample file contains fewer than 200 lines.

Finding Strings

The LOCATE and FIND commands locate specified strings in your file. The NFIND command locates a line that does not contain the specified string.

The LOCATE Command: The LOCATE command looks for a line of text containing a specified string. The format of the LOCATE command is

LOCATE *string*

LOCATE searches text lines beginning immediately after the current line until it finds a line containing *string*. The first line found that contains string is displayed at the terminal and becomes the new current line. If no line containing *string* is found, the pointer is left at the end of the file, and `BOTTOM` is displayed.

string may not contain commas.

For example,

```
PRINT 5
.NULL.
C This program generates the numbers 1 to 10
C and prints the numbers on the terminal screen.
C
      DIMENSION LNUMB (50)
TOP
LOCATE DIMENSION
      DIMENSION LNUMB (50)
```

The FIND Command: The FIND command is a specialized version of the LOCATE command. It searches your file for a line that begins with a specified string. The format of the FIND command is

FIND ***string***

FIND searches lines beginning immediately after the current line for a line that contains *string* beginning in column 1. The first line found that begins with the string becomes the new current line and is displayed at the terminal. If no line beginning with *string* is found, the pointer stops at the end of the file, and BOTTOM is displayed.

string may not contain commas.

For example,

```
FIND C
C This program generates the numbers 1 to 10
FIND 100
100   CONTINUE
```

The NFIND Command: The NFIND command is the inverse of the FIND command; it moves the pointer to the first line below the current line that does not begin with the specified string and displays the line on your terminal screen. The format of the NFIND command is

NFIND ***string***

string may not contain commas.

For example,

```
PRINT 6
.NULL.
C This program generates the numbers 1 to 10
C and prints the numbers on the terminal screen.
C
      DIMENSION LNUMB (50)
        DO 100 I = 1, 10
TOP
NFIND C
      DIMENSION LNUMB (50)
```

Searching on a Specific Column: Using FIND and NFIND, you can also search for a string starting in a column other than column 1 by specifying the column number in the following format:

FIND(*n*) ***string***

where *n* is the column number.

The parentheses () around the *n* are required. Do not include any spaces between FIND and (*n*).

For example,

```
PRINT 12
.NULL.
C This program generates the numbers 1 to 10
C and prints the numbers on the terminal screen.
C
      DIMENSION LNUMB (50)
        DO 100 I = 1, 10
           LNUMB (I) = I
           WRITE (1, 200) LNUMB (I)
200   FORMAT (10X, I5)
100   CONTINUE
      CALL EXIT
      END
TOP
FIND(7) D
      DIMENSION LNUMB (50)
FIND(2) 0
200   FORMAT (10X, I5)
```

You can use the NFIND command in a similar way. The format is

NFIND(*n*) ***string***

For example,

```
NFIND(1) C
      DIMENSION LNUMB (50)
FIND(2) 0
200   FORMAT (10X, I5)
NFIND(2) 0
      CALL EXIT
```

Note

The LOCATE, FIND, and NFIND commands are **case sensitive**; they do not treat uppercase and lowercase letters as equivalent. For example, the command LOCATE *WORD* does not find the string *Word*, because the last three letters of the two strings are not in the same case.

Modifying Text

The commands described in this section alter the text of one or more lines.

The APPEND Command: The APPEND command attaches a specified string to the end of the current line. The format of the APPEND command is

APPEND ***string***

One blank space must separate the command from *string*. Any further blanks are treated as part of the string.

For example,

```
        DIMENSION LNUMB (50)
APPEND , LSTORE (50)
        DIMENSION LNUMB (50), LSTORE (50)
```

The CHANGE Command: The CHANGE command replaces one string in the current line with another string. The format of the CHANGE command is

CHANGE/*string-1*/*string-2*/[G][*n*]

Argument	*Meaning*
string-1	The original string
string-2	The new version of the string

CHANGE allows two optional arguments:

G	If you specify G, CHANGE affects every occurrence of *string-1* on a line. If you do not specify G, only the first occurrence of *string-1* is changed.
n	Specifies the number of lines on which the change is to be made. If *n* is not specified or has a value of 0 or 1, ED makes changes only on the current line. If a value other than 0 or 1 is specified, ED inspects and makes changes on *n* lines, starting at the current line. The pointer is moved to line *n*. If there are fewer than *n* lines after the current line, the pointer stops at the null line at the end of the file, and the message `BOTTOM` is displayed. ED displays all changed lines as well as the last line examined.

Delimiters: The first character after the command word CHANGE is a **delimiter**. ED uses the delimiter to mark the beginning and the end of each of the strings. The slash (/) is customarily used as a delimiter, but you can use any delimiter that is not part of one of the strings. The following example uses slash (/) as a delimiter:

```
        DIMENSION LNUMB (50), LSTORE (50)
CHANGE/DIMENSION/COMMON/
        COMMON LNUMB (50), LSTORE (50)
```

If slash (/) is part of one of the strings, choose a different delimiter. The following example uses the pound sign (#) as a delimiter.

```
          DIMENSION LNUMB (50)/ LSTORE (50)
   CHANGE#/#,#
          DIMENSION LNUMB (50), LSTORE (50)
```

You can use CHANGE to insert characters at the beginning of a line with the following sequence:

CHANGE//string/

For example,

```
LNUMB (50), LSTORE (50)
CHANGE//      DIMENSION /
      DIMENSION LNUMB (50), LSTORE (50)
```

The DELETE Command: The DELETE command deletes a specified number of lines from your file. The format is

DELETE [*n*]

where *n* is the number of lines to be deleted beginning with the current line.

If *n* is positive, then the current line is deleted along with *n*-1 lines after it. If *n* is negative, then the current line is deleted along with *n*-1 lines before it. If *n* is not specified or has a value of –1, 0, or 1, then only the current line is deleted.

ED leaves the pointer at a null line where the last deleted line was located. This enables you to insert new text at this point. The null line is eliminated if you move the pointer to another line in the file. For example,

```
TOP
PRINT 5
.NULL.
C This program generates the numbers 1 to 10
C and prints the numbers on the terminal screen.
C
      DIMENSION LNUMB (50), LSTORE (50)
NEXT -2
C and prints the numbers on the terminal screen.
DELETE
PRINT
.NULL.
TOP
PRINT 4
.NULL.
C This program generates the numbers 1 to 10
C
      DIMENSION LNUMB (50), LSTORE (50)
```

The INSERT Command: The INSERT command inserts a specified new line after the current line. The inserted line becomes the current line. The format of the INSERT command is

INSERT *string*

where *string* is the new line of text to be inserted.

The command word INSERT and *string* must be separated by one space. Any further spaces are treated as part of the string.

For example,

```
      DIMENSION LNUMB (50), LSTORE (50)
        DO 100 I = 1, 10
NEXT -1
      DIMENSION LNUMB (50), LSTORE (50)
INSERT        COMMON LSTART (50)
PRINT 2
        COMMON LSTART (50)
        DO 100 I = 1, 10
```

The IB Command: The IB command inserts a new line before the current line; the inserted line becomes the current line. The format of the IB command is

IB *string*

The command word IB and *string* must be separated by one space. Any further spaces are treated as part of the string.

For example,

```
PRINT 5
.NULL.
C This program generates the numbers 1 to 10
C and prints the numbers on the terminal screen.
C
      DIMENSION LNUMB (50), LSTORE (50)
IB       COMMON LSTART (50)
NEXT -3
C This program generates the numbers 1 to 10
PRINT 5
C This program generates the numbers 1 to 10
C and prints the numbers on the terminal screen.
C
      COMMON LSTART (50)
      DIMENSION LNUMB (50), LSTORE (50)
```

The RETYPE Command: The RETYPE command deletes the current line and replaces it with a specified string. The format of the RETYPE command is

RETYPE *string*

The command word RETYPE and *string* must be separated by one space. Any further spaces are treated as part of the string.

For example,

```
PRINT 3
C Thsi porgarm gennratse hte mumbers 1 too 1999
C and prints the numbers on the terminal screen.
C
POINT 1
C Thsi porgarm gennratse hte mumbers 1 too 1999
RETYPE C This program generates the numbers 1 to 10
PRINT
C This program generates the numbers 1 to 10
PRINT 3
C This program generates the numbers 1 to 10
C and prints the numbers on the terminal screen.
C
```

RETYPE followed immediately by a single space and Return erases the current line and replaces it with a blank line. RETYPE followed by Return alone results in the error message, `BAD RETYPE.`

For example,

```
C This program generates the numbers 1 to 10
C and prints the numbers on the terminal screen.
C
NEXT -1
C and prints the numbers on the terminal screen.
RETYPE                                   You omit blank space after RETYPE.
BAD RETYPE
RETYPE                             You include the blank space after RETYPE.
PRINT
                                          The file contains a blank line here.
NEXT -1
C This program generates the numbers 1 to 10
PRINT 3
C This program generates the numbers 1 to 10

C
```

The OOPS Command: The OOPS command restores the last line changed by any editing command to its previous condition. OOPS does not reverse changes made to several lines at one time. The format of the OOPS command is

OOPS

For example,

```
      DIMENSION LNUMB (50), LSTORE (50)
CHANGE/DIMENSION/COMMON
      COMMON LNUMB (50), LSTORE (50)
OOPS
      DIMENSION LNUMB (50), LSTORE (50)
```

Copying Text To and From Disk Files

The UNLOAD Command: The UNLOAD command copies lines from the file you are editing to another file without deleting the lines from the file you are editing. The format of the UNLOAD command is

UNLOAD ***pathname*** **[*n*]**

Argument	*Meaning*
pathname	The file to which the lines are to be copied. If the file is in your current directory, you can specify the filename alone.

WARNING

If you specify a file that already exists, UNLOAD overwrites it, and its original contents are lost.

n	The number of lines to be copied, including the current line. If *n* is omitted or has a value of 0, 1, or –1, only the current line is copied. When *n* is negative, the current line and *n*–1 preceding lines are copied. The last line to be copied becomes the new current line.

For example,

```
TOP
PRINT 6
..NULL.
C This program generates the numbers 1 to 10
C and prints the numbers on the terminal screen.
C
      DIMENSION LNUMB 50;, LSTORE (50)
        DO 100 I = 1, 10
NEXT -4
C This program generates the numbers 1 to 10
UNLOAD TEMP 3
PRINT 2
C
      DIMENSION LNUMB (50), LSTORE (50)
TOP
PRINT 6
..NULL.
C This program generates the numbers 1 to 10
C and prints the numbers on the terminal screen.
C
      DIMENSION LNUMB (50), LSTORE (50)
        DO 100 I = 1, 10
```

The DUNLOAD Command: The DUNLOAD command copies a specified number of lines from the file being edited to another file and deletes the lines from the file being edited. The format of the DUNLOAD command is

DUNLOAD ***pathname*** **[*n*]**

Argument	*Meaning*
pathname	The file to which the lines are to be moved. If the file is in your current directory, you can specify the filename alone.

WARNING

If the file specified already exists, DUNLOAD overwrites it, and the original contents are lost.

n	The number of lines to be copied and deleted, including the current line. If *n* is not specified or has a value of 0, 1, or –1, the current line is copied and deleted. Negative values of *n* cause the current line and *n*–1 preceding lines to be copied and deleted.

DUNLOAD leaves the pointer positioned at a null line where the deleted lines were located, so that you can insert text at this point. This null line is eliminated when you move the pointer.

For example,

```
TOP
PRINT 6
.NULL.
C This program generates the numbers 1 to 10
C and prints the numbers on the terminal screen.
C
      DIMENSION LNUMB (50), LSTORE (50)
        DO 100 I = 1, 10
NEXT -4
C This program generates the numbers 1 to 10
DUNLOAD TEMP.FTN 3
PRINT 2
.NULL.
      DIMENSION LNUMB (50), LSTORE (50)
TOP
NEXT
      DIMENSION LNUMB (50), LSTORE (50)
PRINT 6
      DIMENSION LNUMB (50), LSTORE (50)
        DO 100 I = 1, 10
BOTTOM
```

The LOAD Command: The LOAD command copies the contents of a disk file into the file you are editing. The material is inserted immediately after the current line. The format of the LOAD command is

LOAD ***pathname***

where *pathname* specifies the file to be inserted. If the file is in the current directory, you can use the filename alone. The file named by *pathname* is not changed by the LOAD operation.

For example,

```
TOP
PRINT 3
..NULL.
      DIMENSION LNUMB (50), LSTORE (50)
        DO 100 I = 1, 10
TOP
LOAD TEMP.FTN
TOP
PRINT 6
.NULL.
C This program generates the numbers 1 to 10
C and prints the numbers on the terminal screen.
C
      DIMENSION LNUMB (50), LSTORE (50)
        DO 100 I = 1, 10
```

Ending and Saving an ED Session

The QUIT, FILE, and SAVE commands end and/or save the current ED session.

The QUIT Command: The QUIT command tells ED to return to PRIMOS command level without saving the ED work file. Any new material you have created and not saved is discarded. If the file you have been editing already exists on disk, the original version remains unchanged. The format of the QUIT command is

QUIT

If you have created or modified a file during the session, ED responds to QUIT with the message

```
FILE MODIFIED, OK TO QUIT?
```

A YES (or Y, YE, O, OK, or Return) response returns you to PRIMOS command level without saving the work file. Any other response provokes the message `PLEASE FILE` from ED. (See the explanation of the FILE command, below.)

If you did not create or modify a file during an ED session, or if you have already saved all new material with the SAVE command (see below), QUIT returns you to PRIMOS command level without any warning messages.

WARNING

If you quit without saving your file, any work you have done during the current editing session is lost.

The FILE Command: The FILE command saves the work file on disk and returns you to PRIMOS command level.

The format for the FILE command is

FILE [*pathname*]

where *pathname* specifies the file in which your work is to be saved. If the file is in the current directory, you can use a filename alone.

If you have been creating a new file (you invoked ED without a pathname or filename), you must specify a filename or pathname.

If you have been editing an old file (you invoked ED with a filename), then you can give the FILE command with or without a *pathname* or *filename*. In these cases

- If you don't specify a pathname or filename, the work file is saved under the original filename. The original version of the file is overwritten.
- If you do specify a pathname or filename, then the work file is saved under that name. The original version of the file remains unchanged.

WARNING

If you specify the name of a file that already exists, FILE overwrites it with the work file, and the file's original contents are lost.

The SAVE Command: The SAVE command saves the contents of the ED work file but does not leave ED or terminate the current session. The format of the SAVE command is

SAVE *pathname*

where *pathname* is the name of the file to which you want the work file copied. If the file is in your current directory, you can give the filename alone.

You can give the SAVE command with or without specifying a pathname or filename. You use the pathname or filename exactly as you do with the FILE command.

SAVE is useful if you make extensive changes to a file, and you want to save your work periodically during the ED session.

WARNING

Because the work file does not exist outside ED, you must use either FILE or SAVE if you want to save your work. If you quit ED without saving your work, then the contents of the work file, including any new text you have added and any changes you have made, are lost. (If you give the SAVE command during a work session, then the work file is saved at that point. If you later quit without a further SAVE or FILE, you lose any changes made since the last SAVE.)

Printing Text Files With SPOOL

Use the SPOOL command to print copies of text files on a printer. Because printing can be relatively slow in computer terms, the SPOOL process is designed to allow you to continue other work with PRIMOS, or even log out, while you wait for your file to be printed. Sending a file to be printed with the SPOOL command is known as **spooling** a file.

The SPOOL command format is

SPOOL $\begin{Bmatrix} \textit{pathname}\ [\textit{options}] \\ \textbf{options} \end{Bmatrix}$

where *pathname* refers to the file you want printed. If the file is in the current directory, you can use the filename alone.

Spool options allow you to choose specific printers, and many other characteristics of your printing job. Because facilities vary from installation to installation, check with your System Administrator to find out which options are available and how to specify them on your system.

If you specify no options, SPOOL prints your file on your system's default printer with all other options set at the default values established for your system. For example, to print the file POETRY.DOC on the default printer, give the command

```
OK, SPOOL POETRY.DOC
```

Ask your System Administrator about the default printer and other default option settings for your installation.

For online information on the SPOOL command and its options, use the –HELP option:

```
OK, SPOOL -HELP
```

The SPOOL Queue

After you make a print request with the SPOOL command, PRIMOS takes over the process of printing your file and returns you to command level so that you can give other commands or even log out. To do this, PRIMOS maintains a list of files waiting to be printed called a **SPOOL queue**. Jobs are put into the queue according to their priority. Jobs of equal priority are queued in the order they are submitted. Short jobs are given higher priority to make the system more efficient. You can also specify that your printing job be deferred to a later time using the –DEFER option of the SPOOL command. (See the section, Deferred Printing, below.) Figure 4-3 illustrates a spool queue.

After you give the SPOOL command, PRIMOS enters your request in the SPOOL queue, and displays the folowing message:

```
Request nn added to queue, x records : pathname
```

nn is a number that identifies the entry in the spool queue. *x* is the number of records in the file. *pathname* is the name of the file you want to print.

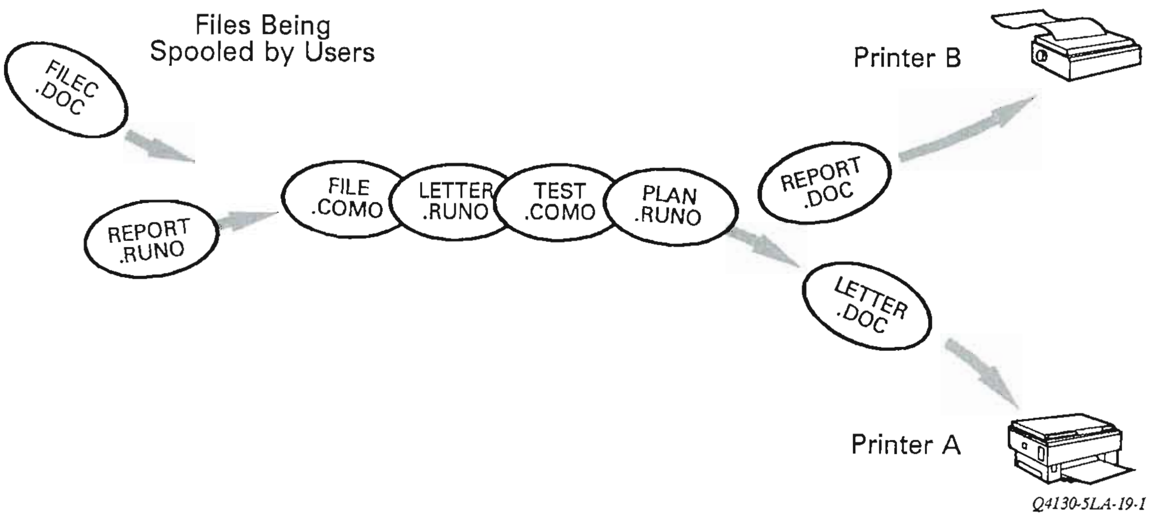


FIGURE 4-3
A SPOOL Queue

Checking the Queue

To check the status of the spool queue, give the SPOOL command with the –LIST option:

```
SPOOL -LIST
```

PRIMOS lists all your job requests in the queue that have not yet been printed. The listing includes the job's number in the queue, time submitted (in 24 hour *hhmm* format), your user ID, the filename, the number of copies requested, and the file size. For example,

```
OK, SPOOL -LIST
[SPOOL Rev. 22.0 Copyright (c) 1987, Prime Computer, Inc.]

System SYSX
Request   Time   User     File          No     Size State
--------- ------ -------- ------------- ------ ---- -----
18        1143   SONYA    BATCH.COMO    1       1
20        1147   SONYA    LIST.COMM     1       2
22        1151   SONYA    FILE3         4       5
```

On some systems you may see information for other users' SPOOL requests as well.

You can get more detailed information with the command SPOOL –LIST –DETAIL.

Canceling a Spool Request

You can cancel a spool request with the –CANCEL option. The format is

SPOOL –CANCEL *n*

where *n* is the file's number on the SPOOL queue. (You can find *n* with the –LIST option, above.)

For example,

```
    SPOOL -CANCEL 18
[SPOOL Rev. 22.0 Copyright (c) 1988, Prime Computer, Inc.]
Request 18 cancelled
```

You can cancel all your spool requests by using the –ALL option.

```
OK, SPOOL -CANCEL -ALL
```

Printing Multiple Copies

You can request several copies of one file by using the –COPIES option. The format is

SPOOL *pathnam* –COPIES *n*

where *n* is the number of copies you want. You can request a maximum of 99 copies. The number of copies requested is displayed in the SPOOL –LIST display in the column headed No.

Deferred Printing

The –DEFER option tells SPOOL not to begin printing the file until the time you specify. This option is useful (and often encouraged by System Administrators) when you have a large file that need not be printed immediately. Such files can be deferred to a time when there is little demand on the printer. You cannot defer the printing of a file for more than 24 hours.

The format of the –DEFER option is

SPOOL ***pathname*** **–DEFER** ***time***

where *time* is given in either 24-hour or 12-hour format. Specify 24-hour format as *hh[:]mm*. You must specify four digits, but the colon (:) is optional. For example, 0200 or 02:00 are both valid; 200 is not. Specify 12-hour as *[h]h[:]mm[AM/PM]*. The colon is optional, but if you include it, you must use four digits. For example, 300PM and 03:00PM are valid; 3:00PM is not.

If you omit *time*, –DEFER defaults to midnight. In networks that span time zones, the computer that holds the queue determines the time.

Choosing a Printer Environment

If you must print your job on a certain type of paper or on a specific printer, you can choose from a set of valid paper or printer types by using the –ATTRIBUTE option.

Note

This option supersedes the –AT, –FORM, and –TYPE options.

The format is

SPOOL –ATTRIBUTE ***name1*** [***name2*** ...]

where each *name* specifies one of the printer or paper types available on your system.

Each system maintains its own list of attribute names, depending on the facilities available. Names for paper types are called **form names**, and names for printers are called **destinations**. In each case, they are strings with a maximum of 16 characters chosen by the System Administrator. Ask your System Administrator what attributes are available and what names they have on your system.

–ATTRIBUTE Examples: Suppose your system allows you to use label paper to print some jobs. If the System Administrator has chosen LABEL as the form name for label paper, you can print the file MAIL_LIST on labels by specifying

```
OK, SPOOL MAIL_LIST -ATT LABEL
```

Your system may allow you to request multipart paper, preprinted forms, and the like with the appropriate form names. Your System Administrator can tell you what is available and what form names to use.

You can similarly request to have your file printed on a specific printer by specifying the appropriate destination. For example, suppose your system has a letter quality printer that your System Administrator has assigned the destination name LQ100. You can request that the file RESUME.DOC be printed on the letter quality printer with the following command:

```
OK, SPOOL RESUME.DOC -ATT LQ100
```

Printing Part of a File

You can print part of a file using the –FROM and –TO options. The format is

SPOOL ***pathname*** **–FROM** ***m*** **–TO** ***n***

SPOOL prints from page *m* through *n*, inclusive. The –FROM value must not be greater than the –TO value. Either –FROM or –TO may be omitted.

Note

If you print formatted documents with SPOOL using the –FROM option, formatting information may be lost if it occurs on pages that are not printed. For example, if a document begins with codes to set a specific font, these codes are not received by the printer if the page that contains them is not printed. As a result, the pages that are printed do not use the requested font.

Receiving Notification

To receive notification on completion of your printing job, use the –NOTIFY option. The format is

SPOOL ***pathname*** **–NOTIFY**

Multiple Options

You can use any or all of the above options except –CANCEL jointly in a single SPOOL command line. For example,

```
OK, SPOOL TEST -COPIES 3 -ATT BLDG.1 -DEFER 2200
[SPOOL Rev. 22.0 Copyright (c) 1988, Prime Computer, Inc.]
Request 48 added to queue, 2 records :  <MFD>MYDIR>TEST
```

This command requests that three copies of the file named TEST be printed at the BLDG.1 printer at 10:00 PM (2200).

You can also cancel printing of all your files spooled on a specific system by using the SPOOL command with the –CANCEL –ALL –ON options. For example, to cancel all of the files that you have queued on system SYSX, use the command

```
OK, SPOOL -CANCEL -ALL -ON SYSX
```

Modifying Options

To modify a file's SPOOL options, use the –MODIFY option:

SPOOL –MODIFY *n options*

File number *n* must already be an entry in the spool queue. You can use any of the SPOOL options listed above. They are substituted for any options you originally specified.

If you want to cancel the –DEFER option you must use the –NODEFER option to remove the –DEFER attribute from a file. In the previous example, the printing of the file TEST, request number 48, was deferred until 10:00 PM (2200). The following command causes TEST to be printed without deferment, according to its place in the SPOOL queue:

```
OK, SPOOL -MODIFY 48 -NODEFER
```

Use the SPOOL –LIST display to verify any changes made with the –MODIFY option.

Other SPOOL Options

Other SPOOL options allow you to choose such characteristics as paper orientation, pagination and page header formats, printer character translation, which paper bin a multibin printer uses, and the like. Some options allow you to SPOOL from files that are still open and to SPOOL directly from your files. (SPOOL normally copies files to its own file area and prints from the copies.) For a complete list of SPOOL options, see the *PRIMOS Commands Reference Guide*.

5

Protecting Your Files and Directories

The file system is available to all logged in users. PRIMOS provides a file protection mechanism called **Access Control Lists (ACLs)** that allows you to control other users' access to your files. This chapter shows you how to use the ACL system.

An Overview of File Protection

PRIMOS controls access to file system objects by associating them with ACLs. An ACL is a list of users and the access rights they are allowed. Whenever you try to carry out any operation on a file system object protected by an ACL, PRIMOS first checks the list to see if you have the required rights. If you do, PRIMOS carries out your task. If not, PRIMOS refuses to carry out the command and displays an error message.

Access Rights

Access rights define the specific activities a user or group of users is permitted to carry out on the protected object. For example, users with **Read access** to a file may read it, users with **Delete access** to a directory may delete objects it contains, and so on. PRIMOS defines nine types of access rights that cover all possible operations on file system objects. These are summarized in Table 5-1 and discussed in detail in the section Types of Access Rights.

Access Control Lists

An Access Control List (ACL) is a list of users and groups of users along with the initials of the access rights granted to each. A typical ACL looks like this:

```
BILL:    DALURWX
$REST:   LUR
```

In this example, user BILL has Delete, Add, List, Use, Read, Write, and Execute rights. $REST, a special designation indicating all other users, have more limited rights: List, Use, and Read.

TABLE 5-1
ACL Access Rights

Symbol	*Right*	*Applies To*	*What Is Permitted*
R	Read	Files	Reading a file
W	Write	Files	Modifying a file
U	Use	Directories	Attaching to directories
L	List	Directories	Listing directory contents
A	Add	Directories	Adding directory entries
D	Delete	Directories	Deleting directory entries
O	Owner	Files and directories	Setting access rights except for P and ALL
P	Protect	Directories	Changing access rights
X	Execute	Local EPFs	Executing a local EPF
ALL	All	Files and directories	All of the above rights
NONE	Files	Files and directories	No access allowed

Protecting Files With ACLs

You protect file system objects by linking them with ACLs. PRIMOS provides several ways to do this:

- You can use a **specific ACL** to protect a specific file system object. PRIMOS treats a specific ACL as an attribute of the object it protects.
- You can create a general version of an ACL, called an **access category**, that you can link with any number of objects in a directory. You can use an access category to provide identical protection to a group of files.
- You can protect large numbers of files, or even your whole file tree, with a single specific ACL or access category. This is possible because your access rights to a parent directory apply to subdirectories lower in the tree. Such protection is called **default protection**.

Setting ACLs

In systems that use ACLs, the System Administrator establishes an ACL for your top-level directory. Except where you specify alternate ACLs, this ACL provides default protection for the other objects in your file tree. If the System Administrator has given you sufficient access rights, you can modify the ACLs on file system objects in your file tree. PRIMOS provides three commands that you can use to list, set, and modify ACL protection: LIST_ACCESS, SET_ACCESS, and EDIT_ACCESS. This chapter shows you how to use these commands to tailor protection of objects in your file system to your needs.

Note

Not all System Administrators choose to allow ACL protection of files. Directory passwords and owner/non-owner access rights provide an alternative to access control lists as a protection system for PRIMOS file system objects. See Appendix F for a discussion of the directory password system and related commands (PASSWD and PROTECT). Appendix F also explains how to convert password-protected directories to ACL-protected directories, and vice versa.

Access Control Lists (ACLs)

An ACL is a list of users along with the access rights granted to each. This section shows you how to list the ACL protecting any file system object and explains how the ACL determines each user's rights.

Listing Access Rights

Use the LIST_ACCESS command to list the ACL protecting any file system object.

The format is

LIST_ACCESS [*pathname*]

where *pathname* refers to the object for which you want to list access rights. If the object is in the current directory, you can use the objectname instead of the pathname. If you do not specify a pathname or objectname, then LIST_ACCESS lists access rights to the current directory.

The following example lists the ACL protecting the current directory:

```
OK, LIST_ACCESS
ACL protecting "<Current Directory>":

       AUTHOR:    ALL
       EDITOR:    ALL
       ADVT:      LUR
       $REST:     NONE
OK,
```

To list access rights for a specified object use the following format:

```
OK, LIST_ACCESS BOOKS>FICTION
ACL protecting "BOOKS>FICTION":
       CAROL:       ALL
       .COMMITTEE:  DALURW
       $REST:       R
OK,
```

Note

You cannot list an ACL with the PRIMOS SLIST command. You must use the LIST_ACCESS command to list an ACL.

The first column of each entry gives the name of a user or group of users, ending with a colon (:). The second column shows the access rights associated with each user or group. An ACL may contain a maximum of 32 of these *user:access right* pairs.

Note

If you change the ACL protecting your current directory, the new ACL takes effect only after you have attached to a directory higher in the file system tree or reattached to your current directory. However, LIST_ACCESS shows the new ACL immediately.

Types of Users

An ACL can identify users in three ways:

- User ID
- Group name
- The special identifier $REST

A **user ID** identifies an individual user. In the example, user CAROL has ALL access rights.

A **group name** is the name of a group of users established by the System Administrator. In PRIMOS, group names begin with a period (for example, .COMMITTEE) to distinguish them from individual user names. For example, if the group name .COMMITTEE represents the user names JANE, FRED, and BARBARA, then each of these users has the rights assigned to .COMMITTEE by the ACL: Delete (D), Add (A), List (L), Use (U), Read (R), Write (W).

$REST specifies the rights granted to all system users not otherwise identified in the ACL. In the example, users other than the members of .COMMITTEE and CAROL are granted Read (R) rights only.

Specifying Rights

The second column of each entry in the ACL shows the rights granted to the user or group listed in the first column. Rights are indicated either by a series of one letter abbreviations or by one of the key words *ALL* or *NONE*. The rights represented by each abbreviation or key word are listed in Table 5-1.

Several abbreviations can be combined in a string, without punctuation, to represent any number of rights. In the example, .COMMITTEE's Delete, Add, List, Use, Read, and Write rights are shown as DALURW. ALL is equivalent to OPDALURWX, all of the rights that exist in the ACL system. NONE means no rights at all.

Overlapping Access Rights

A user who is a member of a group can effectively be listed more than once in a single ACL. PRIMOS determines such a user's rights as follows:

- If a user is listed individually and is also a member of a group listed in an ACL, then the user's rights as an individual take precedence. For example, if CAROL is also a member of .COMMITTEE, she still has ALL access rights, even though other members of .COMMITTEE receive only DALURW rights.
- A user who is a member of more than one group listed in an ACL receives all of the rights granted to each group. Suppose, for example, that your System Administrator creates two user groups with the following members:

.PROJECTA	.PROJECTB
CAROL	CAROL
ANDY	FRED
MARY	ROBIN
TED	
NANCY	

If a file system object is protected by the following ACL,

```
.PROJECTA:        DALURW
.PROJECTB:        LURX
TED:              NONE
```

then CAROL, as a member of .PROJECTA and .PROJECTB, receives DALURWX rights.

Note that TED has no rights. Even though he is a member of PROJECTA, his individual rights as listed in the ACL (in this case NONE) take precedence.

Specific ACLs and Access Categories

ACLs may exist in two forms:

- A **specific ACL** is an attribute of a specific file, directory, or segment directory in your file system tree.
- An **access category** is a separate file system object. You can link an access category to any number of file system objects in the same directory in order to protect them.

Specific ACLs and Access Categories are described in the next two sections.

Specific ACLs

The PRIMOS file system treats a specific ACL as one attribute of the file system object that it protects. The ACL is not itself a separate file system object. For example, when you list the contents of a directory, specific ACLs protecting objects in the directory are not listed.

You can create a specific ACL only for an object that already exists in the file system. When an object protected by a specific ACL is deleted, the ACL is deleted with it.

Note

When you use the COPY command to copy an object protected by a specific ACL, the ACL is not copied with the object unless you use the –ALL option of the COPY command. Objects copied without the –ALL option receive default protection from the directory into which they are copied.

Figure 5-1 shows specific ACLs protecting objects in a file system tree.

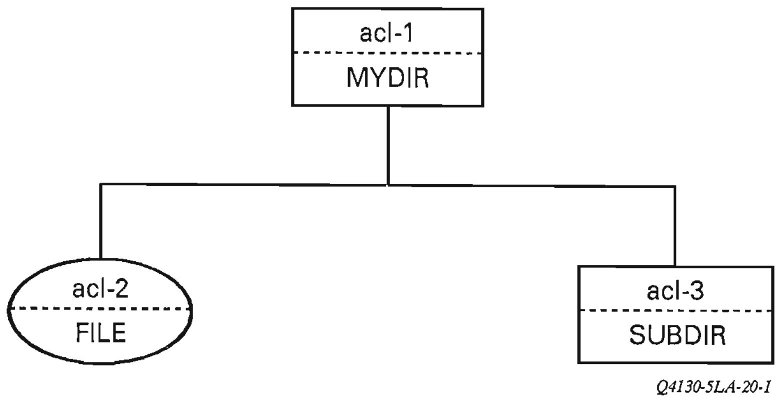


FIGURE 5-1
ACLs Protecting File System Objects

Creating Specific ACLs

To protect an existing file, segment directory, or directory with a specific ACL, use the SET_ACCESS command in the following format:

SET_ACCESS *pathname acl* [–NO_QUERY]

Argument/Option	*Meaning*
pathname	Refers to the file, segment directory, or directory to be protected. If the object is in the current directory, you can use the objectname alone. If the specified object does not exist, SET_ACCESS creates an access category called *pathname*.ACAT as explained below in the section Creating an Access Category. You cannot set an ACL on an access category.
acl	The Access Control List (ACL) specifying access to the object. Type the ACL as in the following example:

```
JANE:ALL .COMMITTEE:DALURW BOB:LUR $REST:NONE
```

The list is a series of pairs of user identifiers and access rights. Enter each pair in the following format:

identifier:access_rights

Argument	*Meaning*
identifier	A user ID, group name, or the identifier $REST
:	A colon separating *identifier* and *access rights* in each pair. Do not put spaces before or after the colon.
access rights	A list of single letter abbreviations for access rights or the designation ALL or NONE. Type lists of letter abbreviations without spaces (for example, DALUR, not D A L U R). You may type the letters in any order.

If the ACL contains more than one pair, separate the pairs with spaces. The ACL may contain as many as 32 pairs, but may not be longer than 160 characters in total, including blanks. The $REST grouping, unless specified on the command line, is automatically given no rights (the designation NONE).

–NO_QUERY PRIMOS normally queries you before replacing the ACL currently protecting the object. The –NO_QUERY option suppresses this query.

Note

In systems that use ACL protection, any object not explicitly protected by a specific ACL or access category is automatically default protected. This means that the new ACL always replaces some previous ACL protection.

The following example shows the use of SET_ACCESS to set a specific ACL:

```
OK, SET_ACCESS BOOKS BOB:DALUR .EDITORS:ALL $REST:LUR
A specific ACL for "BOOKS" already exists.
Do you want to replace it?
```

If you reply with YES or Y, a new specific ACL is created for BOOKS. BOB receives DALUR access rights to BOOKS. Members of the group .EDITORS receive all rights. All other users receive LUR rights.

The following example shows the display you see when you list the ACL:

```
OK, LIST_ACCESS BOOKS

ACL Protecting "BOOKS":
     BOB:         DALUR
     .EDITORS:    ALL
     $REST:       LUR
OK,
```

Modifying a Specific ACL

The EDIT_ACCESS command allows you to modify an existing ACL without replacing it entirely. The format is

EDIT_ACCESS ***pathname acl*** **[-NO_QUERY]**

Argument	*Meaning*
pathname	Refers to an existing file system object for which you want to modify the ACL. If the object is in the current directory, you can give the objectname alone.
acl	Lists the pairs of user identifiers and access rights you want to add or change.
-NO_QUERY	If you use EDIT_ACCESS with an object that is protected by an access category or by default, PRIMOS creates a specific ACL for the object. PRIMOS queries you before creating the new ACL. The -NO_QUERY option suppresses this query.

If you specify a user ID or group ID already listed in the ACL, the user's or group's current access rights are replaced by the new rights you specify. To remove a user or group from an ACL, use a **null access**; list the group or user without any rights identifier (for example, JOHN:).

The following example shows how to use EDIT_ACCESS:

```
OK, LIST_ACCESS REPORTS

"Reports" protected by default ACL (from "<HEADQ>OFFICE"):
         JACK:     LUR
         STEVE:    ALL
         $REST:    NONE
OK, EDIT_ACCESS REPORTS JACK:DALURW JILL:LUR
"REPORTS" is default-protected. Create specific ACL?  YES
OK, LIST_ACCESS REPORTS

ACL protecting "REPORTS":
        JACK:     DALURW
        JILL:     LUR
        STEVE:    ALL
        $REST:    NONE
```

JACK's original rights (LUR) to REPORTS are changed to DALURW. JILL now has LUR access. The original rights of STEVE (ALL) and $REST (NONE) remain unchanged.

WARNING

To change access rights to top-level directories, use EDIT_ACCESS, not SET_ACCESS. If you use SET_ACCESS and fail to include yourself in the access list, you may no longer have any rights at all to your own directory. You can create the same problem with EDIT_ACCESS, but to do so, you must explicitly remove yourself from the ACL. If this happens, see your System Administrator.

Access Categories

An **access category** is an ACL that exists as a separate file system object. You can protect any number of files, directories, or segment directories by linking them to an access category in the same directory.

Access catergories are a convenient way to provide identical protection to a group of related file system objects. Creating a single access category and linking it to several objects is easier than setting the same specific ACL individually on each of the objects. This is especially convenient when the list of users is long or needs frequent adjustment. You can change access to all of the objects at once simply by changing the access category.

Listing Access Categories in a Directory

Like other file system objects, an access category has a name. When you use LD to list the contents of a directory that contains an access category, the access category's name appears along with the names of other objects in the directory. Access category names use the suffix *.ACAT.*

For example, the following directory listing shows two access categories, COVER.ACAT and MEMO.ACAT:

```
OK, LD

<DISK>ACCOUNTING>WORKGROUP (ALL access)
18 records in this directory, 18 total records out of quota of 0.

4 Files.

FINANCES          MEMO1             MEMO2           REPORT

2 Directories

ACCOUNTING        PERSONNEL

2 Access Categories.

COVER.ACAT        MEMO.ACAT

OK,
```

Listing the Contents of an Access Category

Use the LIST_ACCESS command to list the contents of an access category. Give the LIST_ACCESS command with the pathname of the access category you want to list. If the access category is in the current directory you can give the objectname alone.

You need not supply the .ACAT suffix with the access category name unless another object in the directory has the same basename as the access category. For example, consider the two directories in Figure 5-2.

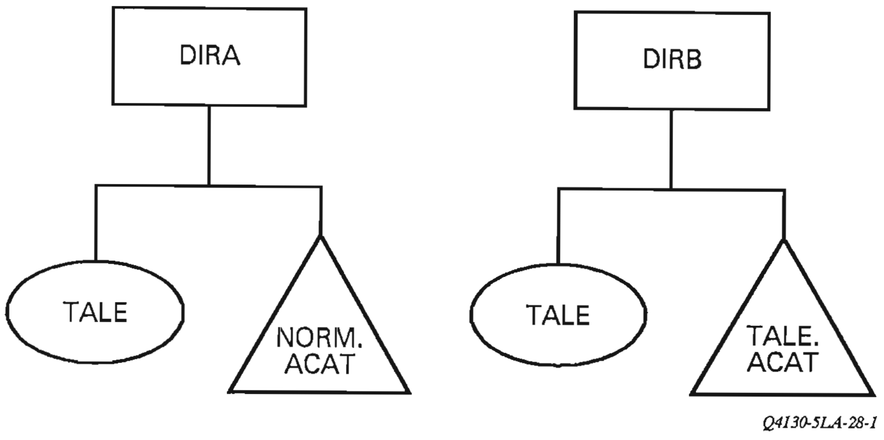


FIGURE 5-2
Listing Access Categories

You can list the contents of NORM.ACAT in DIRA with either of the following commands:

```
OK, LIST_ACCESS NORM
OK, LIST_ACCESS NORM.ACAT
```

To list the contents of TALE.ACAT in DIRB, you must specify

```
OK, LIST_ACCESS TALE.ACAT
```

If you give the command

```
OK, LIST_ACCESS TALE
```

the access rights for the file TALE are displayed. Note the difference between giving the LIST_ACCESS command with the name of an access category and giving it with the name of another type of file system object. In the first case, you see a listing of the contents of the access category itself. In the second case, you see the contents of the ACL protecting the object.

Protecting Objects With Access Categories

Protecting file system objects with access categories is a two step process.

1. If the access category you wish to use doesn't already exist, create it.
2. Link an existing access category to the file system object.

Use a form of the SET_ACCESS command for each step. You can also modify and delete access categories with the EDIT_ACCESS and DELETE commands. The following paragraphs show you how to carry out these operations.

Creating an Access Category: To create an access category give the SET_ACCESS command in the following format:

SET_ACCESS *category-name acl* [-NO_QUERY]

Argument/Option	*Meaning*
category-name	The name of the new access category. If you create the access category in a directory other than current one, use a pathname. If you don't specify the suffix .ACAT, it is appended automatically to the name you give.
acl	Specifies the *identifier:access right* pairs for the access category you are creating.
-NO_QUERY	PRIMOS normally queries you before replacing the ACL currently protecting the object. The -NO_QUERY option suppresses this query.

The following example shows the creation of an access category with SET_ACCESS:

```
OK, SET_ACCESS PROTECT ME:ALL .GROUP:LUR
"PROTECT.ACAT" does not exist.  Create access category? YES
OK,
```

You can list the new access category as follows:

```
OK, LIST_ACCESS PROTECT

Access category "PROTECT.ACAT":
       ME:        ALL
       .GROUP:    LUR
       $REST:     NONE
```

Note

If a file, directory, or segment directory has the same pathname as the access category you want to create, be sure to specify the .ACAT suffix with *category-name* when giving the SET_ACCESS command. Otherwise, SET_ACCESS sets a specific ACL on the existing object instead of creating an access category.

Linking an Access Category to an Object: An access category is not an attribute of any specific file system object. To protect a file system object with an access category, you link the object to the access category using the SET_ACCESS command with the -CATEGORY option. The format is

SET_ACCESS *pathname* -CATEGORY *category-name*

Argument	*Meaning*
pathname	Specifies the object you want to protect. If the object is in the current directory, you can use the objectname alone.
category-name	Specifies the access category to be linked with the object. You don't need to specify the .ACAT suffix. If the access category is in the current directory, you can use the objectname alone.

Remember that the object to be protected and the access category must both be in the same directory.

For example, to protect the file MYFILE with the access category PROTECT.ACAT created in the previous example, use

```
OK, SET_ACCESS MYFILE -CATEGORY PROTECT
```

If you now list access to MYFILE, you see the following display:

```
OK, LIST_ACCESS MYFILE

ACL protecting "MYFILE"
    (from access category "<BOOKS>FICTION>PROTECT.ACAT"):
        ME:       ALL
        .GROUP:   LUR
        $REST:    NONE
OK,
```

The second line of the display shows the pathname of the access category protecting the object.

Figures 5-3 shows how access categories and specific ACLs can protect objects in a directory.

The access category ONE.ACAT protects MEMO and SUBDIR. The access category TWO.ACAT exists, but has not been linked to any file system objects. The file ACCOUNTS is protected by default from YOURDIR, and LETTERS is protected by a specific ACL.

Access categories can be linked only to objects in the same directory. Therefore, when you move or copy an object to a new directory, any link it has to an access category in the original directory is broken. For example, if you copy MEMO to another directory, the new copy loses the protection of ONE.ACAT.

Because an access category is a separate file system object, it continues to exist when you delete any of the objects linked to it. For example, if you delete MEMO and SUBDIR, ONE.ACAT continues to exist as a file system object in YOURDIR, even if you have not linked it to any other files.

Replacing, Modifying, and Deleting Access Categories

Replacing the Contents of an Access Category: If the access category already exists, the SET_ACCESS command replaces the category's existing access list with the new access list specified on the command line. The format is identical to that for creating new access categories:

SET_ACCESS ***category-name acl*** [-NO_QUERY]

You need not include the .ACAT suffix when you specify the access category name.

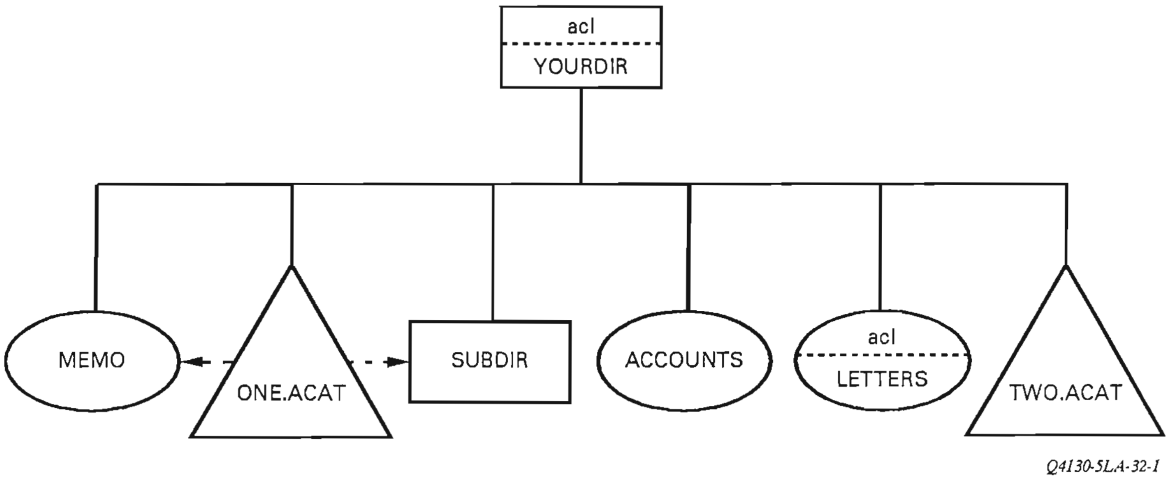


FIGURE 5-3
Access Categories in a Directory

The following example shows you how to change the contents of the access category PROTECT.ACAT:

```
OK, SET_ACCESS PROTECT ME:ALL $REST:LUR
"PROTECT.ACAT" is an existing access category.
Do you want to replace it?  YES
OK,
```

PROTECT.ACAT now contains the following ACL:

```
    ME:        ALL
    $REST:     LUR
```

Modifying an Access Category: Use EDIT_ACCESS to modify an existing access category without replacing it entirely. The procedure is the same as using EDIT_ACCESS to modify a specific ACL except that you supply the pathname of an access category. For example,

```
OK, LIST_ACCESS TEST.ACAT

Access category "TEST.ACAT":
        MOZART:    ALL
        SALIERI:   LUR
        $REST:     NONE
```

```
OK, EDIT_ACCESS TEST.ACAT SALIERI: $REST:LUR

OK, LIST_ACCESS TEST.ACAT

Access category "TEST.ACAT":
        MOZART:    ALL
        $REST:     LUR
  OK,
```

Note

Using SET_ACCESS or EDIT_ACCESS to set or change access to an object protected by an access category is different from using these commands to create or modify an access category. If you set or change access to an object protected by an access category, the access category itself is not changed, although it ceases to protect the object. Instead, SET_ACCESS and EDIT_ACCESS establish a new specific ACL for the object.

Deleting an Access Category: To delete an access category, use the DELETE command, as for any other file system object:

DELETE ***category-name***

If you delete an access category, any objects protected by the access category lose the access category's protection, reverting to default protection.

Default Protection

ACLs, whether they are specific ACLs or access categories, provide protection for file system objects in two ways:

- ACLs protect the object(s) to which they are directly linked. In the case of a specific ACL, this means the single file system object to which the ACL is linked. In the case of an access category, this means any objects to which the access category is linked.
- ACLs that are linked to parent directories protect objects contained in those directories or lower down in the file system tree, unless those objects are protected by other ACLs. This is called **default protection.**

Default protection applies only to objects that are not directly linked to a specific ACL or an access category. When you create a specific ACL for an object or link an object to an access category, the object is no longer default protected.

When you carry out any operation on an object that is default protected, PRIMOS treats the object exactly as if it were directly linked to the ACL that is providing the default protection. For example, if the default ACL gives you Read access, then you have Read access to the default-protected object.

Listing Default Protection

When you use LIST_ACCESS to list access rights to a file system object that is default protected, you see a display like the following:

```
OK, LIST_ACCESS

"<Current directory>" protected by default ACL (from "<MUSIC>CHOPIN"):
        PIANO:           ALL
        .COMPOSERS:      LUR
        $REST:           NONE

OK,
```

The first line of the display shows the pathname of the object that provides the default protection.

The following diagrams and discussion illustrate how default protection can affect objects throughout a tree structure.

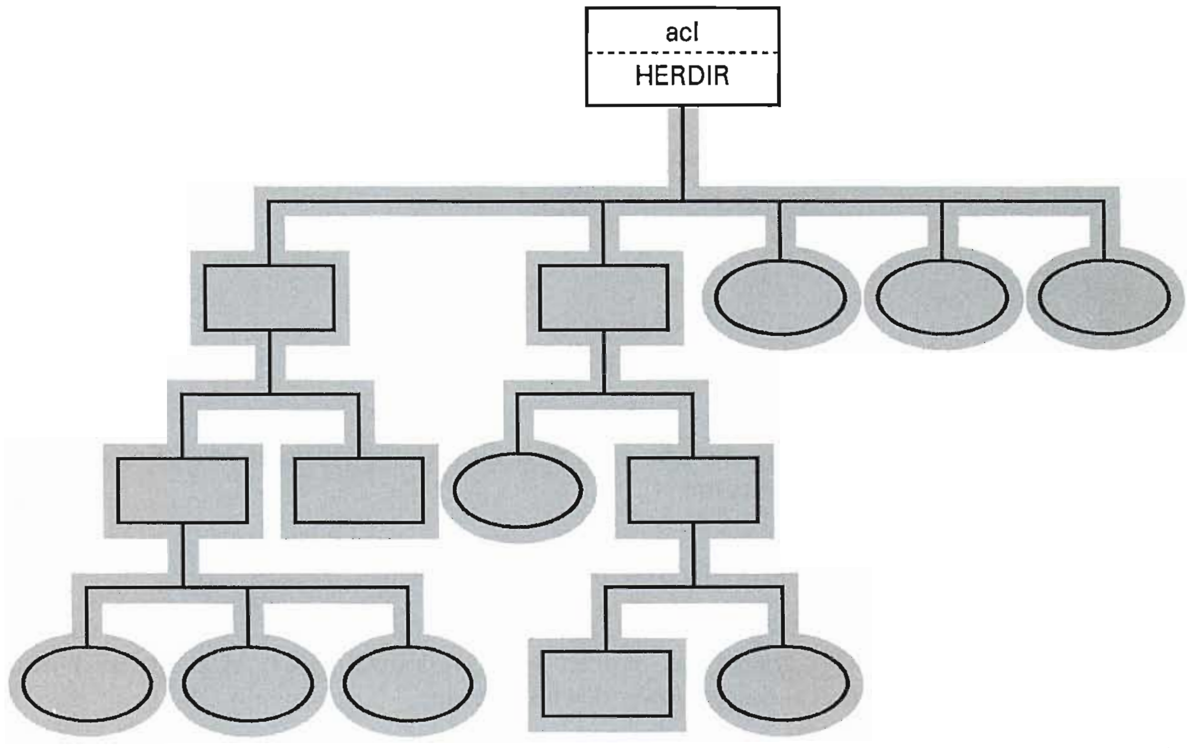
Providing Default Protection With Specific ACLs

In Figure 5-4, the specific ACL on top-level directory HERDIR provides default protection for a whole file system tree. The access rights defined by the ACL for HERDIR apply to all of the file system objects lower down in the tree.

Note

A specific ACL set on a directory does *not* automatically provide default protection for subdirectories that are already protected through the password protection system. Passworded directories must be explicitly converted to ACL protection. If you create new subdirectories under the ACL protected directory, they are automatically protected by default. For an explanation of the password protection system and conversion to ACL protection, see Appendix F.

Objects in a default protected tree need not use the default protection. You can provide different protection for objects in a default protected tree by setting different ACLs on them. Whatever protection you set lower down in the tree takes precedence over the default protection.



Q4130-5LA-22-1

FIGURE 5-4
A Directory Tree Under Default Protection

Figure 5-5 shows how specific ACLs modify default protection.

In Figure 5-5, specific ACLs modify default protection in the following ways:

- The specific ACL on OURDIR provides default protection only for FILE1.
- FILE2 and SUBDIR are each protected by specific ACLs that take precedence over the default protection provided by the ACL on OURDIR.
- FILE3 and FILE4 are now default protected by the specific ACL on SUBDIR.

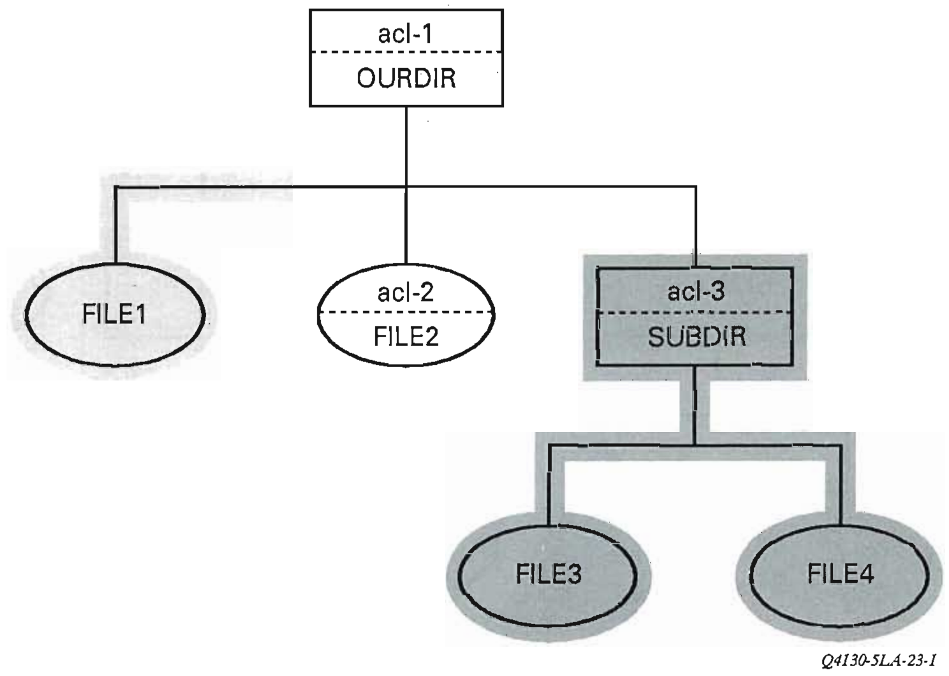


FIGURE 5-5
Default Protection and Specific ACLs

Providing Default Protection With Access Categories

Access categories can also provide default protection. If you link an access category to a directory, this access category automatically provides default protection for all file system objects lower in the tree unless you explicitly protect them with other ACLs.

Figure 5-6 shows default protection provided by access categories.

In Figure 5-6, specific ACLs and access categories provide default protection in the following ways:

- The specific ACL on YOURDIR provides default protection for FILE.X only.
- The access category GUARD.ACAT is linked with SUBDIR1 and SUBDIR2 and takes precedence over the default protection provided by the ACL protecting YOURDIR.
- The protection that GUARD.ACAT provides to SUBDIR2 also extends by default to FILE2, FILE3, and FILE4.
- The access category OTHER.ACAT is linked to FILE1 so that FILE1 is not subject to default protection by GUARD.ACAT.

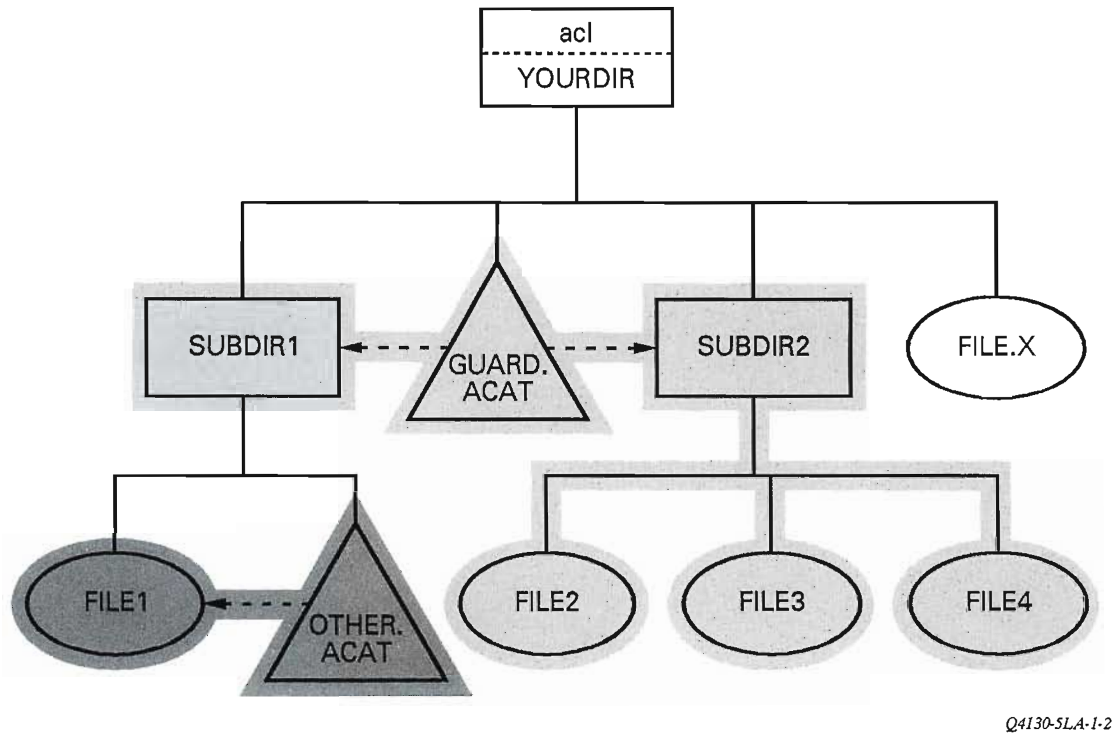


FIGURE 5-6
Default Protection and Access Categories

Figure 5-7 illustrates how all three protection mechanisms, specific ACLs, access categories, and default protection, can exist in the same directory tree.

In Figure 5-7, the three protection mechanisms provide default protection in the following ways:

- The specific ACL on YOURDIR provides default protection to SUBDIRA and FILEA.
- The specific ACL on FILEB takes precedence over the default protection.
- The access category A.ACAT is linked to SUBDIRB, taking precedence over the default protection provided by the ACL on YOURDIR.
- A.ACAT protects FILEC and SEGDIR by default.
- The specific ACL on SUBDIRC takes precedence over the default protection provided by A.ACAT. It also provides default protection for FILED.

Returning to Default Access

You can remove the protection provided by either a specific ACL or an access category and return an object to default protection using the SET_ACCESS command in the following format:

SET_ACCESS *pathname*

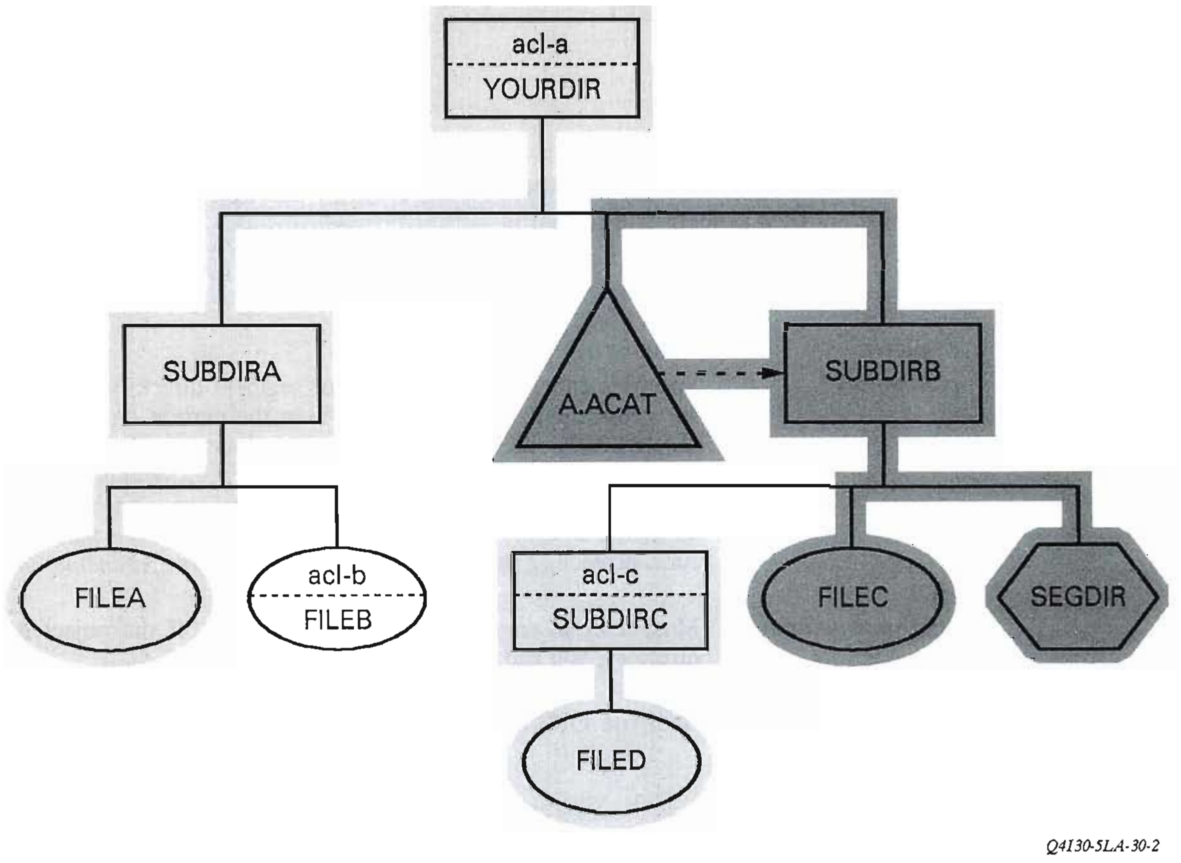


FIGURE 5-7
Default Protection, Specific ACLs, and Access Categories

where *pathname* refers to a file, directory, or segment directory from which you wish to remove protection. For objects in the current directory you can use the objectname alone. You cannot remove protection from an MFD. For example,

```
OK, LIST_ACCESS MEMO

ACL protecting "MEMO":
        BOB:      ALL
        $REST:    DALURW
OK, SET_ACCESS MEMO
OK, LIST_ACCESS MEMO

"MEMO" protected by default ACL (from "<BOOKS>ACCOUNTS"):
        JEAN:     ALL
        $REST:    NONE
OK,
```

The original rights to the file MEMO (`BOB:ALL $REST:DALURW`) have been removed and replaced with the default access rights for the directory <BOOKS>ACCOUNTS (`JEAN:ALL $REST:NONE`).

Matching Access Rights

You can use SET_ACCESS with the –LIKE option to make access rights to one object match rights to another object. The format is

SET_ACCESS ***pathname* –LIKE *reference-pathname***

Argument	*Meaning*
pathname	Specifies either a file, directory, or segment directory you want to protect or an access category. If the object is in the current directory, you can use the objectname alone. If the object is a file, directory, or segment directory, a specific ACL is established to protect it. The ACL grants the same rights as the ACL protecting the object specified by *reference-pathname*. If the object is an access category, the access control list it contains is made identical with the ACL protecting the object specified by *reference-pathname*.
reference-pathname	Must refer to an existing file system object. If the object is in the current directory, you can use the objectname alone.

For example, suppose that file OUTLINE is protected by the following ACL:

```
MARY:     ALL
.GROUP:   LUR
$REST:    NONE
```

You can establish an identical specific ACL to protect the file REPORT using

```
OK, SET_ACCESS REPORT -LIKE OUTLINE
```

Who Can Set ACLs

In systems that use ACLs, the System Administrator sets ACL protection on each user's top-level directory. This protection provides default protection for your files unless you set other ACLs within your file tree. You can set other ACLs if your System Administrator gives you Protect (P) or Owner (O) access to your top-level directory. These rights are described in detail below, in the section Types of Access Rights.

By changing ACLs you can tailor access to your file system tree in order to share access to some objects and limit access to others. Similarly, you can give some users more rights than others to your files. These points are illustrated in Figures 5-8 and 5-9.

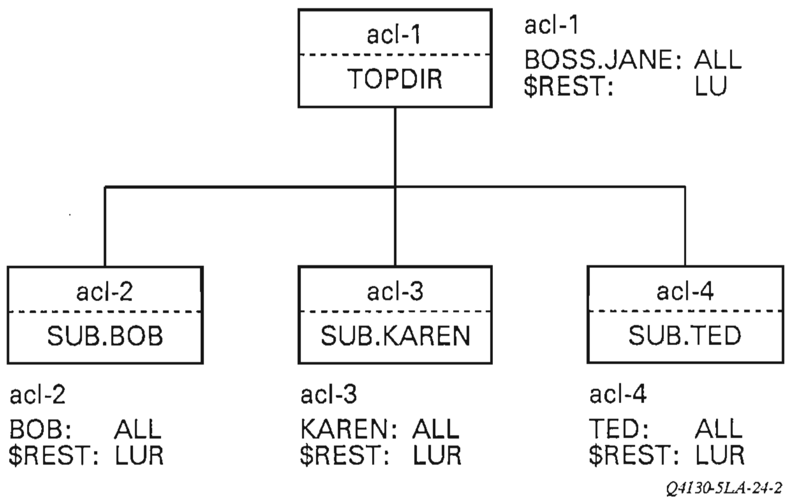


FIGURE 5-8
ACLs in a Directory Tree

In Figure 5-8, BOSS.JANE has ALL rights to TOPDIR. She has granted ALL rights in SUB.BOB, SUB.KAREN, and SUB.TED to BOB, KAREN, and TED, respectively. The other users have more limited rights. For example, BOB has ALL rights to SUB.BOB and can change rights to it at will. However, as a member of $REST, BOB has only List, Use, and Read rights to SUB.KAREN and SUB.TED, and may not change these rights.

Bob may grant rights to subdirectories lower in his own branch of the tree, as illustrated by Figure 5-9.

In Figure 5-9, BOB has granted ALL rights to JOE and to himself in SSUB.JOE and ALL rights to JIM and himself in SSUB.JIM. Other users have no rights.

You may wonder if BOSS.JANE has been excluded from rights in the subdirectories. As the rights have been distributed, BOSS.JANE is included in the $REST category in the ACLs protecting the subdirectories. In the case of the directories SUB.BOB, SUB.KAREN, and SUB.TED, she has LUR rights only. In SSUB.JIM and SSUB.JOE, she has no rights at all.

However, BOSS.JANE's rights to TOPDIR make it possible for her to gain other rights lower down in the tree. P rights to a directory always give you the right to change the ACLs on file system objects in the directory. Therefore, BOSS.JANE can change her rights to SUB.BOB, SUB.KAREN, and SUB.TED. If she grants herself P rights to SUB.BOB, she can also change her rights to SSUB.JIM or SSUB.JOE. The next section gives more details on P rights.

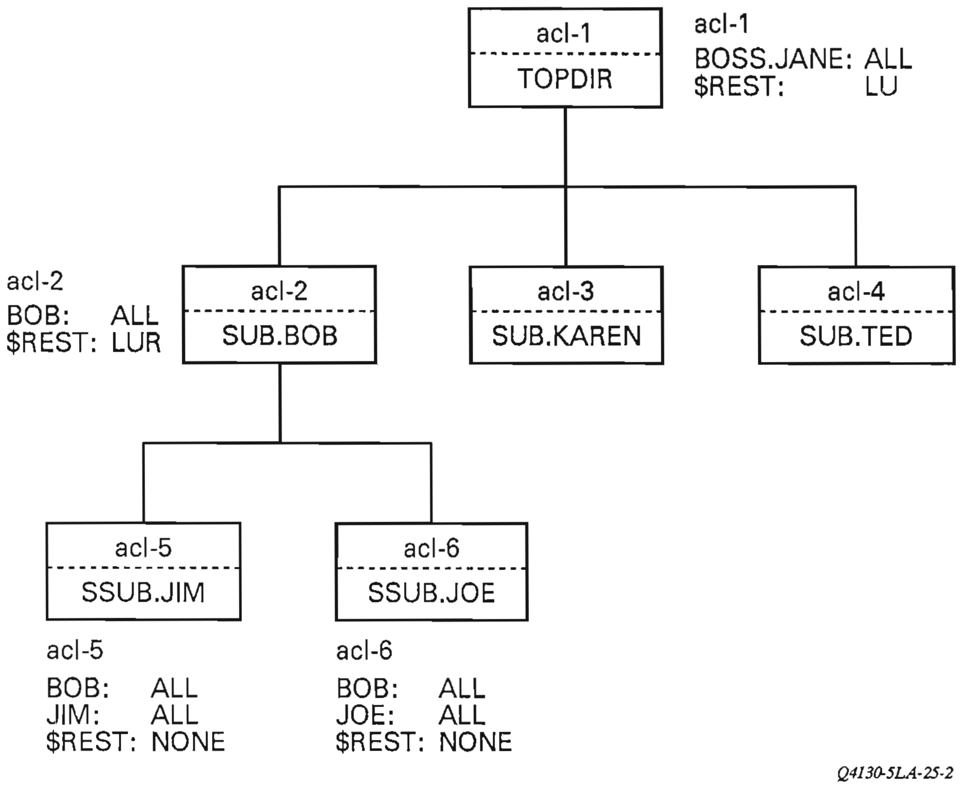


FIGURE 5-9
More ACLs in a Directory Tree

Types of Access Rights

The ACL system defines nine types of access rights. These are summarized in Table 5-1 and described in the following sections.

Some access rights apply to files and some apply to directories. However, rights to directories can have important effects on your ability to carry out operations on objects in those directories. Delete access, for example, applies to directories only. You cannot apply delete access to an individual file. Yet having delete access to a directory allows you to delete files in the directory.

Access rights applied to a parent directory may also give you the ability to carry out operations on members of the directory that you cannot carry out on the directory itself. For example, Delete access to a directory gives you the right to delete objects within the directory, but not the directory itself. On the other hand, Use rights to a directory give you the right to attach to the directory itself, but not to any subdirectories.

Some access rights apply only to files and not to directories. You can still set such access rights on directories. When you do this, the rights apply by default to objects lower down in the file system tree.

Protect (P)

Protect is the most powerful of all access rights. If you have protect access to a directory, you may create or modify the specific ACL protecting the directory. You can also create or modify any ACLs protecting the objects that are immediate members of the parent directory, even if those member objects are protected by ACLs that do not include P access.

P access is meaningful only when applied to directories. Even if the ACL protecting a file specifies P access, you can change the ACL only if you have P access to the parent directory. The only exception to this rule occurs in the case of access categories. If the rights listed in an access category include P rights for you, then you can change the access category even if you don't have P rights to its parent directory. (You cannot, however, create a new access category unless you have P rights to the directory in which you want to create it.)

Note that P access, unlike other access rights, allows you to carry out certain operations (changing ACLs) on both a directory and its members. If you have P access to a parent directory you can change ACLs for both the parent directory and member objects.

As with other rights, if you have P access to a directory, you also have default P access to any subdirectories. Don't confuse this default P access to member directories with your rights to set ACLs on member objects. The difference is illustrated by Figure 5-10.

Suppose you have P access to the directory PARENT, and PARENT in turn contains the directory CHILD. CHILD is protected by an ACL that gives you only DALURW rights, that is, no P rights. Because you have P access to PARENT, you can change the ACL protecting CHILD even though you don't have P rights to CHILD.

The difference between being able to change the ACL on CHILD and actually having P rights to CHILD is significant. For example, CHILD itself contains the directory GRANDCHILD. Because you don't have P rights to CHILD, you can't set ACLs on GRANDCHILD unless you first give yourself P rights to CHILD.

Owner (O)

Owner rights are a more restricted version of P rights. If you have O rights to a directory, you can create or change the specific ACL protecting that directory. You are allowed to establish an ACL containing any rights except P rights. (You can, however, remove P rights from an ACL that includes them.)

Unlike P rights, O rights can also be applied to individual files and segment directories. If you have O rights to a file or segment directory, you can change the ACL protecting it whether or not you have O or P rights to its parent directory. You can change an access category if the access category itself grants you O rights.

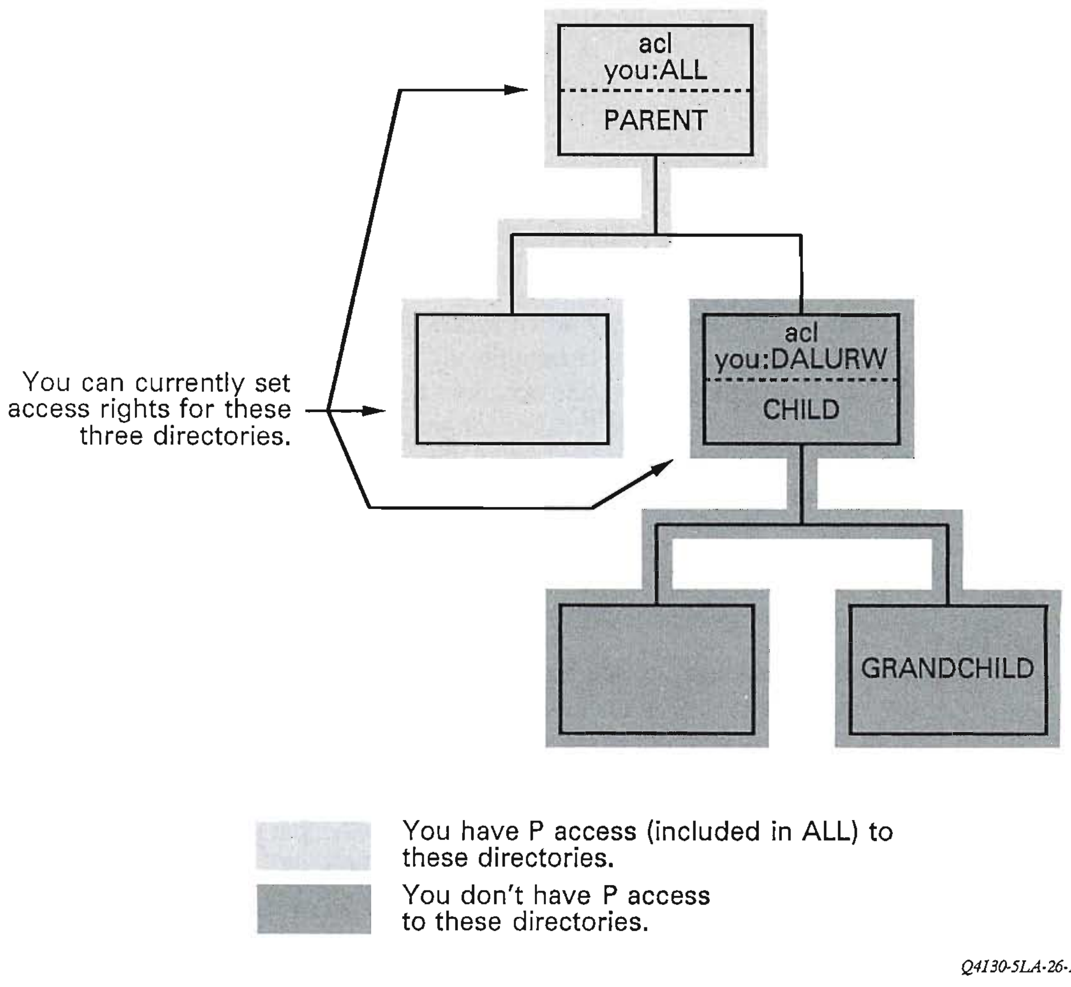

acl
you:ALL
PARENT
acl
you:DALURW
CHILD
GRANDCHILD
You can currently set access rights for these three directories.
You have P access (included in ALL) to these directories.
You don't have P access to these directories.
Q4130-5LA-26-1

FIGURE 5-10
Protect Access

Unlike P rights, O rights to a directory do not automatically give you the right to set or change ACLs on objects that are members of the directory. Of course, if you have O rights to a parent directory, then you have O rights by default to the objects it contains as long as they are not protected by ACLs that exclude you from O rights. You can change the ACLs on any objects to which you have this default protection.

Delete (D)

Delete access allows you to delete file system objects. Delete access applies to directories. Delete access to a directory gives you the right to delete the files and directories it contains. It does not give you the right to delete the directory itself. In order to delete EPFs (Executable Program Formats) and segment directories, you also need Write (W) access to those objects.

Note

Since Delete access applies to whole directories and not to individual files, you can mark important files as delete-protected with the SET_DELETE command (explained in Chapter 3).

Even if you have Delete access to a directory, you cannot delete member directories if you do not have the right to delete objects lower down in the file tree. This situation is illustrated by Figure 5-11.

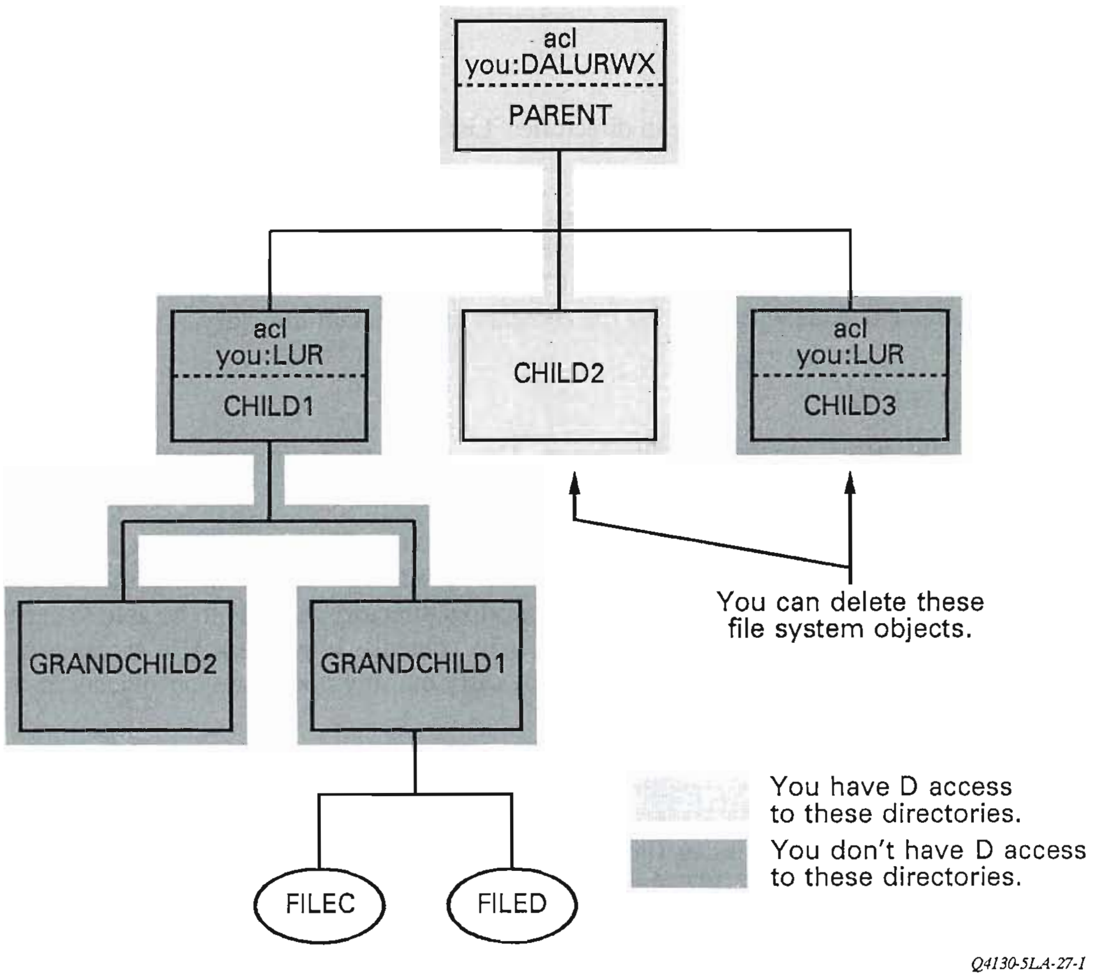

acl
you:DALURWX
PARENT
acl
you:LUR
CHILD1
CHILD2
acl
you:LUR
CHILD3
You can delete these file system objects.
GRANDCHILD2
GRANDCHILD1
FILEC
FILED
You have D access to these directories.
You don't have D access to these directories.
Q4130-5LA-27-1

FIGURE 5-11
Delete Access

Your Delete access to PARENT gives you the right to delete CHILD2 and CHILD3, even though CHILD3 is protected by an ACL that does not give you Delete access. You cannot delete CHILD1, however. You do not have delete access to GRANDCHILD1, so you cannot delete the files it contains, FILEC and FILED. Since these files are below CHILD1 in the directory tree, they prevent you from deleting CHILD1.

Add (A)

Add access applies to directories. If you have Add access to a directory, you can create new file system objects in the directory. When you create objects in a directory, they have default protection from the ACL protecting the directory until you create an ACL that otherwise protects them.

Note

Unless you have Write (W) access to an object you cannot modify it. Therefore, if you create an object in a directory to which you have A but not W access, you cannot modify the object later unless you change the ACL protecting it.

List (L)

List access applies only to directories. List access to a directory allows you to list the contents of the directory.

When you have L access to a parent directory, you can always list the names of all the file system objects that it contains. Even if the parent directory contains directories or other objects that are protected by ACLs that deny you L access, you can still list the names of those objects when you list the contents of the parent directory.

Use (U)

Use access applies to directories. Use access to a directory allows you to attach to the directory and use the directory name in a pathname.

Use access is the most restricted form of access. If you have only Use access to a directory you cannot carry out such operations as adding files or listing the directory's contents. Conversely, you must have Use access to a directory in order to be able to carry out any other operations involving the directory or its contents. If you lack Use access to a directory, and thus cannot attach to it, you cannot carry out any operations on objects in the directory or lower down in the file tree.

Read (R)

Read access applies to files. If you have Read access to a file, then you can read the file's contents (using SLIST, for example) or copy it (using COPY, for example). Read access also includes the capabilities of Execute (X) access. If you have Read access to an EPF, you can execute the EPF.

Read access to a directory is not meaningful in itself. (You need L access to list the contents of a directory.) However, if you have Read access to a directory, then you have Read access by default to all file system objects lower down in the tree unless they are protected by ACLs that exclude you from R access.

Write (W)

Write access applies to files. If you have W access to a file you can modify it by rewriting it. If you execute a program that rewrites a file, you must have W access to the file that is to be rewritten. For example, you can use a text editor to read a file into memory and edit it if you have R access. However, you must have W access to the file in order to have the text editor overwrite the old version of the file with the newly edited version. You must have W access to an EPF or segment directory in order to delete it.

W access to a directory is not meaningful in itself. (In order to modify a directory by adding and deleting members you must have A and D access.) However, if you have W access to a directory, you also get W access by default to all file system objects lower down in the tree unless they are protected by ACLs that exclude you from W access.

Execute (X)

Execute access applies to local EPFs. If you have X access to a local EPF, you can execute it, but you cannot read or copy it unless you have R access. To execute a remote EPF, you must have R access.

X access is not directly meaningful when applied to directories. However, if you have X access to a directory, then you have default X access to EPFs lower down in the file system tree unless other ACLs exclude you from X access.

ALL

The ALL designation grants all rights to a user. Specifying OPDALURWX instead of ALL on the command line grants the same rights as ALL.

Note

If your system has upgraded to Rev. 21.0 or later from a previous version of PRIMOS, objects that had ALL rights before have been changed to PDALURWX rights. To add the new O right, you can use EDIT_ACCESS to either add O rights or restore the ALL designation, which now includes O rights.

NONE

The NONE designation denies all access. Unless you explicitly specify otherwise, the $REST group is automatically given NONE as access rights when you create ACLs.

Additional Information on Access Rights

Keep the following points in mind when you set access rights:

- You can set access rights to your current directory. If you list your access rights with LIST_ACCESS, your new rights are listed. However, PRIMOS does not recognize these rights until you have attached to a directory higher in the file system tree or reattached to your current directory. Before that, PRIMOS treats your rights as if they were unchanged.

- Many of the above descriptions specify that rights are meaningful only when they are applied to directories. PRIMOS allows you to set these rights for other types of file system objects using SET_ACCESS and EDIT_ACCESS. However, when PRIMOS executes commands that affect these file system objects, it ignores any access rights that are not meaningful.
- Some PRIMOS operations require specific combinations of access rights. For example, to change the name of a file system object using CNAME, you must have both D and A access rights to the directory that contains the object. To copy a file using COPY you must have R access to the object being copied, and A access to the directory in which it is to be placed. When copying a directory, you must have R access to all the files lower in the file system tree and A access to the directory in which the copied directory is to be placed.

Priority ACLs

Sometimes the operator or System Administrator needs to control all access to the system (during backups, for example). For this reason, the operator or System Administrator may temporarily override any user-defined ACL by creating a priority ACL. A **priority ACL** defines access for the entire disk. When a priority ACL is active on a disk, the LIST_ACCESS command displays the priority ACL's contents following the normal ACL listing.

For example,

```
OK, LIST_ACCESS

ACL protecting "<Current directory>":
    JOHN:     ALL
    .GROUP:   ALL
    $REST:    LUR

Priority ACL in effect for "<Current directory>":
    .ADMINISTRATORS:  ALL
OK,
```

The LIST_PRIORITY_ACCESS Command

The LIST_PRIORITY_ACCESS command allows you to read the contents of the priority ACL on any MFD. This is useful if a priority ACL prevents you from getting access to an MFD. In such a case, you cannot list the priority ACL's contents with the LIST_ACCESS command. The format of the LIST_PRIORITY_ACCESS command is

LIST_PRIORITY_ACCESS *diskname*

For example,

```
OK, LIST_PRIORITY_ACCESS FOREST
Priority ACL on partition "<FOREST>":
    DEER:    ALL
    $REST:   NONE
OK,
```

If no priority ACL exists on the disk, PRIMOS displays the following message:

```
Priority ACL not found.  <FOREST> (list_priority_access)
ER!
```

6 Command-line Features

Several **command-line features** allow you to carry out many repetitious operations with a single command. These features allow you to

- Enter several commands on one line
- Use iteration to repeat commands
- Use wildcard characters to refer to several related objectnames in a single command argument
- Use treewalking to search for related files throughout a file system tree
- Use name generation characters to create several related objectnames with a single command
- Suppress some features of command line interpretation

The *PRIMOS Commands Reference Guide* provides more detailed information on each feature discussed in this chapter. The following chapters also deal with more advanced command line features. Chapter 7 shows you how to use the command-line editor EDIT_CMD_LINE (ECL) to correct errors and repeat commands. Chapter 8 explains the ABBREV command, which you can use to create single word abbreviations for frequently used command lines. Chapter 8 also tells you how to use global variables to replace frequently used character strings in your command lines.

Multiple Commands

You can give several commands on one line if you separate the commands with semicolons (;). For example, the command line

```
OK, ATTACH MYDIR; LD
```

attaches you to MYDIR and lists the contents of the directory.

Errors in Multiple Commands

In a line containing several commands, one command may include an error. If this happens, PRIMOS still tries to execute the remaining commands on the line. For instance, if PRIMOS is unable to attach you to MYDIR in the above example, it still lists the contents of whatever directory you are currently attached to.

Iteration

You can use **iteration** to specify a list of two or more objects as arguments for one command. The list must be enclosed in parentheses. Items in the list must be separated by one or more blanks or by commas. For example, the command

```
OK, DELETE (PREFACE INTRO CHAP1)
```

deletes the three files PREFACE, INTRO, and CHAP1 from your current directory.

You can replace more than one argument in a command with an iteration list. Each list must be enclosed in parentheses, and the lists must be separated by one or more blanks or by commas. For example, the command

```
OK, COPY (ACCT PAID DUE) (ACCOUNTS PAYMENT PASTDUE)
```

copies the file ACCT as ACCOUNTS, PAID as PAYMENT, and DUE as PASTDUE.

You can use an iteration list as one part of a command argument. For example, the command

```
OK, COPY BOOK>(INTRO CHAP1 CHAP2) NEWBOOK>(FRONT SEC1 SEC2)
```

copies the three files INTRO, CHAP1, and CHAP2 from the directory BOOK into the directory NEWBOOK, and names the new files FRONT, SEC1, and SEC2, respectively.

Further, you can use two iteration lists separated by a period (.) in one argument. Each list specifies one component of the argument. This creates a series of arguments from all possible combinations consisting of one element from the first list and one element from the second list. For example, the command

```
OK, DELETE (YOUR,MY,HIS).(MEMO,DRAFT)
```

deletes YOUR.MEMO, MY.MEMO, HIS.MEMO, YOUR.DRAFT, MY.DRAFT, and HIS.DRAFT from your current directory.

You cannot combine more than two iteration lists in a single argument.

Wildcards

You can use **wildcard** characters to specify a group of objectnames with a single command argument. Each character stands for one or more elements in an objectname. The four wildcard characters are listed in Table 6-1.

TABLE 6-1
Wild Characters

Character	*Function*
@@	Replaces any number of characters in any number of components within a filename or directory name.
@	Replaces any number of characters within one component of a filename or directory name. Stops matching at the period (.) that separates components.
+	Replaces a single character, except a period (.).
^	Negation character; matches all names that do not match the rest of the wildcard name. If you use the caret (^), it must be the first character in the wildcard name. Only one wildcard negation character can be used in a command line.

The following examples show you how to use each character. Consider the following directory:

```
<MYMFD>WORK>BOOK (ALL access)
15 records in this directory, 15 total records out of quota of 0.

13 Files.

C1                C2                C3                C4
NC1               NC2               NC3               NC4
PROGRAM.BIN       PROGRAM.FTN       PROGRAM.LIST      PROGRAM.RUN
STATS.LIST
```

You can select various files to be deleted by using the DELETE comand with different wildcard characters. For example, if you are currently attached to the directory BOOK, then the command

```
OK, DELETE @@
```

deletes all the files from the directory (without deleting the directory itself).

If you are currently attached to WORK, you can also delete all of the files from BOOK with the command

```
OK, DELETE *>BOOK>@@
```

Note

When you use wildcard characters in this way, they must be in the last element of the pathname. You use wildcard characters in intermediate positions for treewalking file tree searches. Treewalking is explained in the next section. You cannot use wildcard characters as the first element of a pathname. For example, @@>BOOK is illegal.

If you are currently attached to BOOK, you can delete all the files with single component names (C1, C2, C3, C4, NC1, NC2, NC3, NC4) using

```
OK, DELETE @
```

You can delete only files with two component names (PROGRAM.FTN, PROGRAM.LIST, PROGRAM.BIN, PROGRAM.RUN, and STATS.LIST) using

```
OK, DELETE @.@
```

If you want to delete only the PROGRAM.*suffix* files, use

```
OK, DELETE PROGRAM.@
```

If you want to delete just the files C1, C2, C3, C4, use

```
OK, DELETE C+
```

If you want to delete everything *except* C1, C2, C3, C4, use

```
OK, DELETE ^C+
```

Note

Commands that take more than one argument can include wildcards in only one argument. For example, you cannot use COPY PROGRAM.@ OLDPROG.@. When you want to create a new set of objectnames using wildcards, you must use name generation characters, discussed below in the section Name Generation.

Wildcards With Iteration Lists

You can use wildcards and iteration lists in the same command argument. For example, the command

```
OK, DELETE @.(BIN LIST)
```

deletes all the files in the current directory that are suffixed with .BIN or .LIST. In the above example, this command line deletes PROGRAM.BIN, PROGRAM.LIST, and STATS.LIST.

Wildcard Options

Wildcard options are used in a command line that contains wildcard characters to restrict the set of files selected. Wildcard options can restrict the selection in two ways:

- By selecting only file system objects of particular types (directories, files, segment directories, or access categories)
- By selecting only file system objects last modified before or after a particular date and time

When you give a command with a wildcard argument, PRIMOS can also query you to verify that you want the command carried out on each file selected. Certain options enable or disable this verification.

Table 6-2 lists the wildcard options. You can use them anywhere in the command line after the command. For example, to delete all files from the current directory that were last modified before April 15, 1987 at 11 p.m. use the command

```
OK, DELETE @@ -BEFORE 87-04-15.23:00:00
```

Wildcard options are explained in more detail in Chapter 4 of the *PRIMOS Commands Reference Guide*.

TABLE 6-2
Wildcard Options

Option	*Selects*
–ACCESS_CATEGORY	Access categories.
–ACCESSED_AFTER *date.time*	Objects last accessed on or after *date.time*.
–ACCESSED_BEFORE *date.time*	Objects last accessed before *date.time*.
–AFTER *date.time*	Objects last modified on or after *date.time*.
–BACKEDUP_AFTER *date.time*	Objects saved by BACKUP on or after *date.time*.
–BACKEDUP_BEFORE *date.time*	Objects saved by BACKUP before *date.time*.
–BEFORE *date.time*	Objects last modified before *date.time*.
–CREATED_AFTER *date.time*	Objects created on or after *date.time*.
–CREATED_BEFORE *date.time*	Objects created before *date.time*.
–DIRECTORY	MFDs, directories and subdirectories.
–FILE	SAM or DAM files.
–MODIFIED_AFTER *date.time*	Same as –AFTER.
–MODIFIED_BEFORE *date.time*	Same as –BEFORE.
–NO_VERIFY	PRIMOS executes the command for all selected objects, without asking for verification of each object.
–RBF	ROAM files.
–SEGMENT_DIRECTORY	SAM or DAM segment directories.
–VERIFY	PRIMOS lists each object selected and asks whether to execute the command for that object. (Default)

Note

In Table 6-2, the format of *date.time* can be any of the following:

yy-mm-dd.hh:mm:ss	***'dd mon yy hh:mm:ss'***
mm/dd/yy.hh:mm:ss	***'dd mon yy.hh:mm:ss'***

The two formats on the right must be enclosed in single quotation marks; *mon* stands for a month's first three characters (JAN, FEB, for example). All the other letters represent one-digit or two-digit numbers. The *hh* field uses 24-hour notation. Omitted date fields are replaced by current date information; omitted time fields are replaced by zeros.

Using the LD Command With Wildcards

The LD command as introduced in Chapter 3, without arguments or options, lists the contents of the current directory. You can use a more general form of the LD command with wildcards. The format is

LD[*pathname*] [*objectname1...objectname14*] [*options*]

Argument	*Meaning*
pathname	The pathname of an object you want to list. PRIMOS displays the object's parent directory header followed by the objectname. If you give only an objectname, PRIMOS looks for the object in your current directory. If the object specified by pathname doesn't exist, PRIMOS replies with the message `No entries selected`. You can list several objects by using wildcards in the last element of the pathname. The LD command without arguments is thus equivalent to typing `OK, LD @@`
objectname1. . .	You can specify a maximum of 14 more objectnames to be listed in the same directory as the object specified by *pathname* (that is, fifteen objectnames in total). Normally, you specify these objectnames with wildcards to select groups of objects.
options	Various options allow you to select the format and level of detail of the listing. These are explained in the *PRIMOS Commands Reference Guide*.

Examples: Suppose that you use the LD command without options to list all the files in your current directory:

```
OK, LD

<MATH11>YOURDIR>STATS (ALL access)
10 records in this directory, 10 total records out of quota of 0.

8 Files.

ANOVA.BIN          ANOVA.FTN          INPUT.RUN          SAMPLE.COMO
STATS.BIN          STATS.FTN          STATS.RUN          TEST.BIN
```

You can list all of the files with the .BIN suffix, as follows:

```
OK, LD @.BIN

<MATH11>YOURDIR>STATS (ALL access)
10 records in this directory, 10 total records out of quota of 0.

3 Files.

ANOVA.BIN         STATS.BIN         TEST.BIN
```

In the sample directory, all the filenames have two components. A more general version of the above command, which lists filenames with any number of components, is

```
OK, LD @@.BIN
```

You can list all files with the .BIN and .RUN suffixes, as follows:

```
OK, LD @@.BIN @@.RUN

<MATH11>YOURDIR>STATS (ALL access)
10 records in this directory, 10 total records out of quota of 0.

5 Files.

ANOVA.BIN         STATS.BIN         TEST.BIN
INPUT.RUN         STATS.RUN
```

To list objects in a different directory, give the full pathname. For example, the command

```
OK, LD MYDIR>MAIL
```

lists the name of the directory MAIL (if it can be located in MYDIR):

```
<PUBL2>MYDIR (ALL access)
32 records in this directory, 57 total records out of quota of 0.

1 Directory.

MAIL
```

You can list the contents of MAIL as follows:

```
OK LD MYDIR>MAIL>@@

<PUBL2>MYDIR>MAIL (ALL access)
4 records in this directory, 4 total records out of quota of 0.

3 Files.

LETTER1           LETTER2           LETTER3
```

Treewalking

You can use wildcards to make a command act on designated objects throughout a file system tree. This is called **treewalking**.

You specify a treewalking pattern by using a wildcard name in an intermediate position within a pathname. For example,

```
OK, LD BOOKS>@@>FICTION
```

You cannot use wildcard characters in the first position of the pathname, and you can only use wildcard characters in a single intermediate position.

The final position of the pathname may also contain a wildcard name. For example,

```
OK, LD BOOKS>@@>FICTION>POE.@
```

Treewalking searches for objects throughout the file system tree below the specified starting directory. A file system tree beginning with a specified directory is called a **directory tree**. Figure 6-1 shows an example of a directory tree. Treewalking searches the directory tree below the starting directory.

When PRIMOS makes a treewalking search, it breaks the wildcarded pathname at the first wildcard character to make a pattern for finding objects. The search locates all objects whose pathnames match the two halves of the specified pathname. For example, if you are attached to the directory BOOKS in Figure 6-1, then the command

```
OK, LD BOOKS>@@>@@
```

causes PRIMOS to search for all objects in the directory tree whose pathnames match the pattern

```
BOOKS> . . . @@>@@
```

In other words, the search matches all objects whose pathnames begin with BOOKS> and end with a pattern that matches @@>@@.

PRIMOS adds whatever intermediate directory names are necessary to match the pattern. For the directory tree shown in Figure 6-1, the search matches the following patterns:

```
BOOKS>@@>@@
BOOKS>FICTION>@@>@@
BOOKS>FICTION>RECENT>@@>@@
```

In this case, the LD command lists the contents of all the directories in the directory tree below BOOKS.

To list the contents of all directories in the directory tree below FICTION, use the command

```
OK, LD BOOKS>FICTION>@@>@@
```

To list all objects with the objectname suffix .DOC in the directory tree below FICTION, use the command

```
OK, LD BOOKS>FICTION>@@>@@.DOC
```

The listed objects are shaded in Figure 6-1.

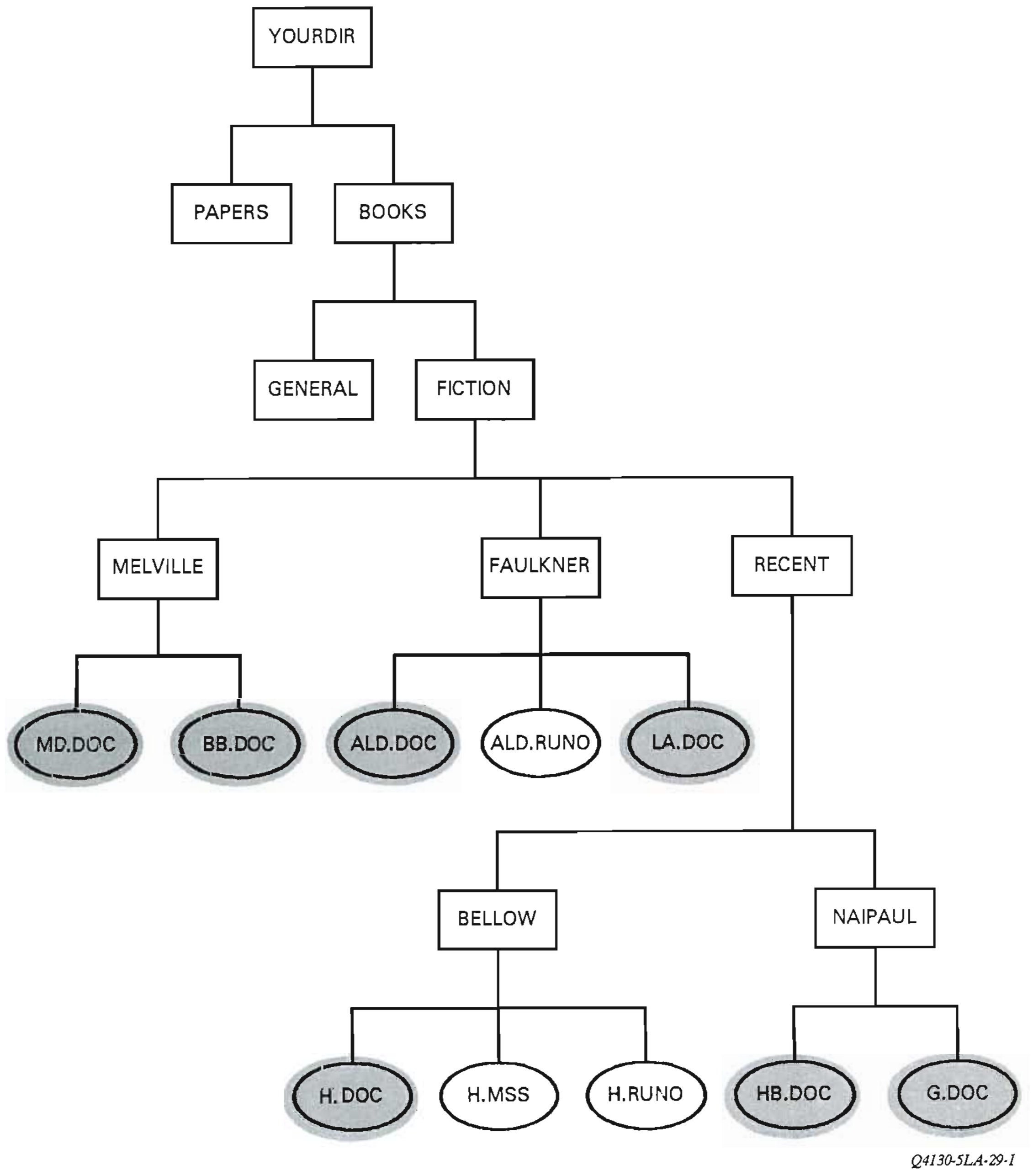


FIGURE 6-1
Sample Directory Tree

To list the contents of all the directories whose names begin with B, use the command

```
OK, LD BOOKS>B@>@@
```

In this case, the command lists the contents of the directory BELLOW.

Treewalking Options

Treewalking normally examines the entire directory tree beginning with the first directory below the starting directory. You can narrow the search or reverse the order of search by using **treewalking options**. The treewalking options are listed in Table 6-3.

TABLE 6-3
Treewalking Options

Option	*Meaning*
-WALK_FROM *n*	Carries out the treewalk in directories at levels greater than or equal to *n*. The default is WALK_FROM 2, which executes the command at the first directory under the starting directory. For execution in the starting directory, specify -WALK_FROM 1.
-WALK_TO *n*	Carries out the treewalk in directories at levels less than or equal to *n*.
-BOTTOM_UP	Carries out the treewalk in bottom-up order, starting at the deepest level and proceeding to the higher levels.

Figure 6-2 shows how PRIMOS determines directory levels. For example, the command

```
OK, LD ORCHARD>@@>@@ -WALK_FROM 1 -WALK_TO 3
```

lists the contents of the directory ORCHARD, its subdirectories, and their subdirectories.

You can reverse the order in which the directories are listed using the following command line:

```
OK, LD ORCHARD>@@>@@ -WALK_FROM 1 -WALK_TO 3 -BOTTOM_UP
```

Figure 6-2 shows where a bottom-up treewalk starts in the sample directory tree.

Name Generation

You use wildcard characters to find a series of objectnames in a directory. Similarly, you can use **name generation characters** to create a series of new objectnames. Name generation can also allow you to find objects with similar names more precisely than wildcards.

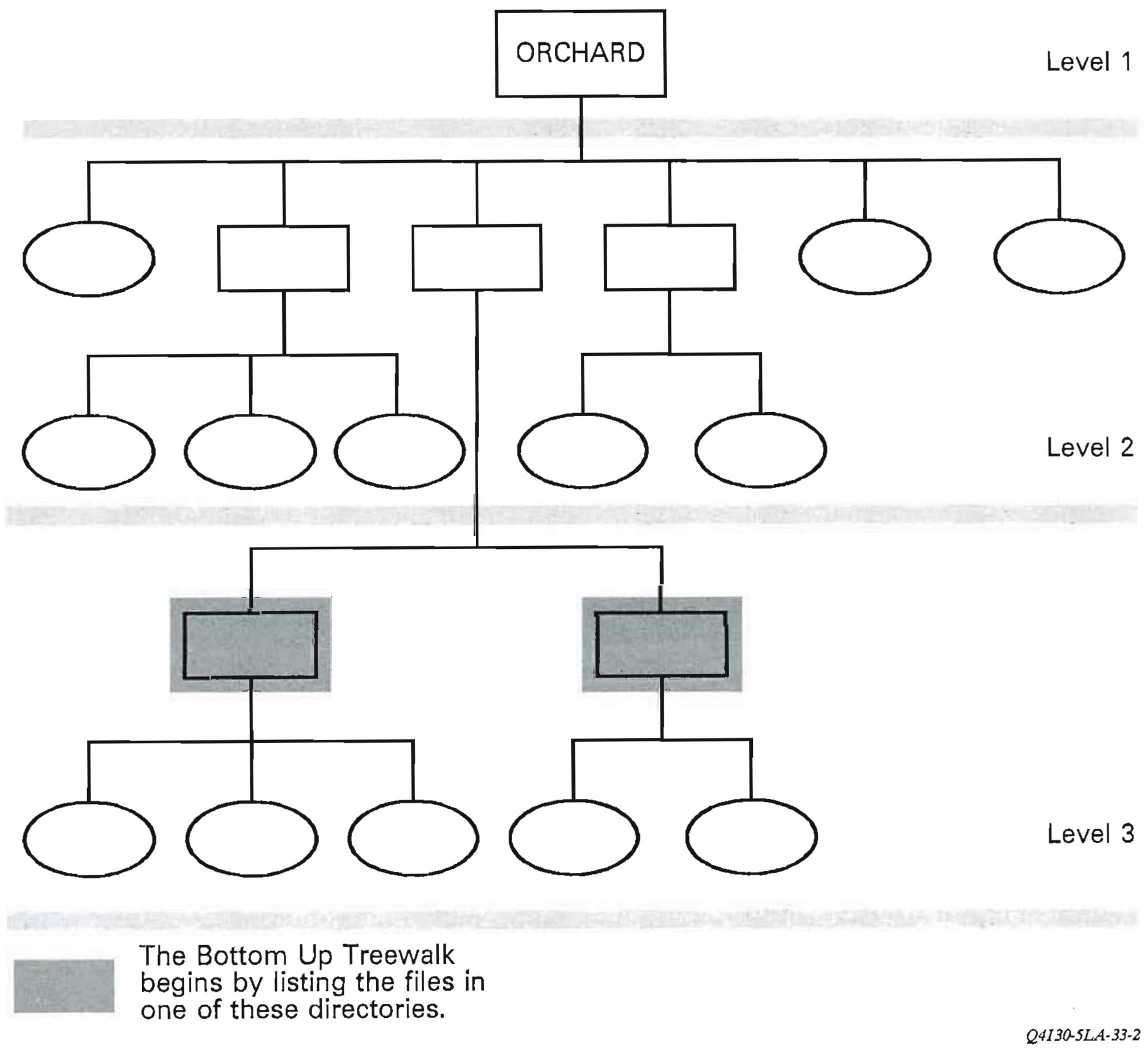

ORCHARD
Level 1
Level 2
Level 3
The Bottom Up Treewalk begins by listing the files in one of these directories.
Q4130-5LA-33-2

FIGURE 6-2
Treewalking Directory Levels

To use name generation you need

- A **source pathname**, from which to create new names. This is usually the first argument in a command line.
- A **generation pattern**, which tells PRIMOS how to create new pathnames from the source pathname. This is usually the last element of one or more subsequent arguments in the command line. Generation patterns are made up of name generation symbols and literal strings.

The name generation characters are listed in Table 6-4.

TABLE 6-4
Name Generation Symbols

Symbol	*Meaning*
=	Copies a single component from the source name to the generated name
==	Copies one or more components from the source name to the generated name
^=	Excludes a single component of the source name from the generated name
^==	Excludes one or more components of the source name from the generated name
literal-string	Replaces a component of the source name with *literal-string*
+*literal-string*	Adds the component given by *literal-string* to the generated name

Note

Only one double equal sign (==), with or without the caret (^) symbol, may appear in a name generation pattern.

Examples

Suppose that you want to copy the files PROGRAM.FTN, PROGRAM.BIN, PROGRAM.LIST, and PROGRAM.RUN from the current directory to the directory UPDATE. You can use name generation to create names for the copies.

To keep the same names, use the command line

```
OK, COPY PROGRAM.@ UPDATE>==
```

To copy and rename the files NEWPROG.*suffix*, use

```
OK, COPY PROGRAM.@ UPDATE>NEWPROG.=
```

This command line creates the files NEWPROG.FTN, NEWPROG.BIN, NEWPROG.LIST, and NEWPROG.RUN.

If you want to generate names with more components than the source name, use the plus sign (+). For example, to copy the same files and rename them OLD.PROGRAM.*suffix*, use

```
OK, COPY PROGRAM.@ UPDATE>+OLD.==
```

This command line creates the files OLD.PROGRAM.FTN, OLD.PROGAM.BIN, OLD.PROGRAM.LIST, and OLD.PROGRAM.RUN.

You can eliminate a component with the caret (^) symbol. For example, if you want to copy and rename the files OLD.PROGRAM.*suffix* as OLD.*suffix*, use

```
OK, COPY OLD.PROGRAM.@ UPDATE>=.^=.=
```

This command creates the files OLD.FTN, OLD.BIN, OLD.LIST, and OLD.RUN.

Selecting File System Objects

You can use name generation to find a series of objects with similar names as well as to create new files. For example,

```
OK, LD CHAP.A =.B =.C =.D
```

lists the files CHAP.A CHAP.B CHAP.C, and CHAP.D (if they are in the current directory). Name generation can be a more precise way of selecting files than wildcards. For example, you can also list these files with

```
OK, LD CHAP.@
```

However, this command line also lists any other files with name CHAP.*suffix*.

For a more detailed discussion of name generation, and many more examples, see the *PRIMOS Commands Reference Guide.*

Syntax Suppression

You can use the tilde character (~) at the beginning of a command line to suppress certain features of command line interpretation. Such **syntax suppression** is especially useful when you are working with abbreviations and global variables, introduced in Chapter 8. Chapter 8 includes an example of syntax suppression.

7

Command-line Editor

The PRIMOS command-line editor, **EDIT_CMD_LINE (ECL)**, lets you edit, save, and redisplay PRIMOS command lines. You do not need to retype entire command lines if you enter wrong characters by mistake. With ECL you can

- Move the cursor to the beginning or end of the line, or move it a specified number of characters or words in either direction
- Delete a specified number of characters or words, or delete everything to the right of the cursor
- Edit and insert command line text
- Locate, redisplay, and edit as many as 200 of the most recent commands executed during a terminal session
- Save commands in a file for subsequent reexecution
- Redisplay deleted command line text
- Expand partially entered pathnames
- Perform all of the above functions on the local system or on a remote system across a network

You can perform most of these functions with just one or two keystrokes. The keystrokes required are very similar to those used within the EMACS editor. However, you do not have to know EMACS to use ECL.

The first part of this chapter shows you how to get started with ECL. The second part explains more advanced ECL features. A summary of ECL subcommands and descriptions of ECL options appear at the end of the chapter.

Getting Started

You invoke ECL and control many of its features with the EDIT_CMD_LINE command. The format is

EDIT_CMD_LINE ***options***

This chapter introduces several basic options, and briefly lists many more. For a complete discussion of the ECL command and all its options, see the *PRIMOS Commands Reference Guide*.

To begin using the command-line editor, enter the ECL command with the –ON option:

```
OK, ECL -ON
```

After you give the ECL command, you can use the features of the command-line editor whenever you are at PRIMOS command level.

Note

The command-line editor is not yet available in most subsystems. When you are working with a subsystem that does not include ECL, you must use the editing features normally available with the subsystem, even if you are using ECL at PRIMOS command level. For example, if you invoke ED after invoking ECL, you still must use the PRIMOS default editing procedures on the ED command lines you type.

To disable ECL, enter

```
OK, ECL -OFF
```

For convenience, you may want to add the ECL –ON command line to your login file so that ECL is in effect as soon as you log in.

By default, ECL uses whatever PRIMOS ready and error prompts are in effect when you invoke ECL. You can have ECL display different prompts. See the section, Customizing Your ECL Prompts.

This section covers ECL basics:

- ECL command conventions
- Aborting ECL commands
- Moving the cursor on the current line
- Deleting characters and lines
- Refreshing the current command line
- Redisplaying the previous command line
- Searching for previously entered text

ECL Command Conventions

All ECL commands begin with either Esc or a combination of Ctrl and another key.

For example, the command to move the cursor to the beginning of a line is Ctrl A. To execute this command, press Ctrl and A simultaneously.

With Esc commands, you must press and release Esc before you press the following characters. For example, the command for a reverse search is Esc R. To execute this command, first type Esc, then type R.

You can precede ECL commands by a numeric count that represents the number of times to repeat that command. To repeat an ECL command, press Esc and then press the digits for the number of times you want to repeat the next command. (Esc 3 tells ECL to perform the next ECL command entered three times.) Refer to the section, Repeating ECL Commands, for more information.

The position of the cursor determines which part of the line is affected by the ECL command entered. Some ECL commands affect text to the left of the cursor, some affect text to the right of the cursor, and some affect only the character over which the cursor is positioned. In sample command lines, the underscore character (_) represents the cursor position.

Moving the Cursor on the Current Line

The most basic ECL cursor commands move the cursor to the beginning or end of a line, and forward or backward by one or more characters. Table 7-1 shows the commands to perform these functions. Individual sections describing each command follow the table.

TABLE 7-1
Cursor-moving Commands

ECL Command	*Position*	*Name*
Ctrl A	Beginning of line	**begin_line**
Ctrl E	End of line	**end_line**
Ctrl B	Back one character	**back_char**
Ctrl F	Forward one character	**forward_char**

Note

Each ECL command has a command name. Command listings show the command name in the last column. For example, in Table 7-1, the command to position the cursor at the beginning of the line is called **begin_line**. The section, Extended Commands, later in this chapter, shows you how you can invoke ECL commands using their command names.

Moving the Cursor to the Beginning of the Line: To move the cursor to the beginning of the line, press Ctrl A. For example, suppose that you have just entered the command line below. The underscore character represents the cursor, shown at the end of the command line.

```
OK, *>STATUS_
```

What you really wanted to do was attach to the STATUS directory. Instead of using the PRIMOS erase or kill characters and then retyping the entire command line, use the ECL Ctrl A command.

```
OK, *>STATUS_
Ctrl A
OK, *>STATUS
```

Ctrl A positions the cursor on the first character of the line, in this case, the asterisk. Now enter the PRIMOS ATTACH command and press Return to execute the command line.

```
OK, ATTACH *>STATUS
```

Notice that the cursor remains positioned on the asterisk. When you enter characters on a line, any existing characters to the right of the cursor are shifted to the right. ECL automatically wraps as many as 158 characters of command line input.

Also note that when you press Return, PRIMOS receives and processes all characters on the command line, regardless of the cursor's position.

Moving the Cursor to the End of the Line: To move the cursor to the end of the line, press Ctrl E. Suppose that in the previous example, you wanted to edit a file in the STATUS directory. The cursor is currently in the middle of the command line, as shown below.

```
OK, ATTACH *>STATUS
Ctrl E
OK, ATTACH *>STATUS_
```

Now you can enter an additional command, using a semicolon (;) to separate the two commands.

```
OK, ATTACH *>STATUS; ED JUN.30_
```

Unlike Ctrl A, which positions the cursor *on* the first character of the command line, Ctrl E positions the cursor *after* the last character on the line. This allows you to add text to the end of the line.

Moving the Cursor Backward: To move the cursor backward on the command line without deleting text, press Ctrl B. If you press Ctrl B once, the cursor moves back one character. If you hold the two keys down instead of pressing them briefly, the cursor continues to move backward until you release the keys.

Take a look at the following command line.

```
OK, SLIT MANNERS_
```

To change SLIT to SLIST, press Ctrl B until the cursor is positioned on the T. Then enter S, as shown below.

```
OK, SLIT MANNERS_
Ctrl B                                    Hold the keys down.
OK, SLIT MANNERS
S
OK, SLIST MANNERS
```

Notice that the character entered appears to the left of the character at which the cursor is positioned. The text after the new character is shifted to the right to make room.

Moving the Cursor Forward: Press Ctrl F to move the cursor forward one or more characters. As with all Ctrl commands, if you hold the two keys down for several seconds, the cursor continues to move until you release the keys.

Deleting Characters and Lines

ECL observes erase and kill characters specified with the PRIMOS TERM command unless you indicate otherwise using the ECL –NO_OBEY_ERKL option. This option is documented in the options list at the end of this chapter. ECL also has its own commands that delete characters and lines, shown in Table 7-2.

TABLE 7-2
Delete Commands for Characters and Lines

ECL Command	*Deletion*	*Command Name*
Ctrl D	Character that cursor is on	**delete_char**
Ctrl H	Previous character	**rubout_char**
Ctrl K	Rest of line (from cursor to end)	**kill_line**
Ctrl U Ctrl W	Entire line	**kill_region**

Notes

Ctrl H works the same way as the PRIMOS erase character (defined with the TERM –ERASE command) and the Backspace key.

Ctrl U Ctrl W works the same way as the PRIMOS kill character. See the section, Deleting Regions, for more information on Ctrl W.

With ECL you also can delete words in either direction, as well as regions of text. These other delete commands are described in the section, Advanced ECL Commands, later in this chapter. The section, Redisplaying the Previous Command Line, includes an example of Ctrl D.

Refreshing the Current Command Line

Pressing Ctrl L redisplays, or refreshes, the current ECL command line. Ctrl L is especially useful if you receive a message from a phantom or another user before you have completed an ECL function. This may disrupt the display of your command line (although it does not alter the version of the line that ECL maintains internally). You can restore the display with Ctrl L.

Redisplaying the Previous Command Line

Press Ctrl Z to redisplay the most recently executed command line. Ctrl Z is particularly useful if you misspell something on a command line just executed. The following example illustrates this point.

```
OK, COPY JAKE>CARD>ENGINE.PROB CARS>COMPLAINTS>JAKE.ENGINE
```

In this command line the pathname of the file being copied is incorrect; the CARD directory should be the CARS directory. Consequently, when you press Return, PRIMOS displays an error message. The following steps show how to correct this error.

1. Press Ctrl Z to redisplay the command line, with the cursor positioned after the last character on the line.

```
OK, COPY JAKE>CARD>ENGINE.PROB CARS>COMPLAINTS>JAKE.ENGINE_
```

2. Press Ctrl B to move the cursor back to the D in CARD.

```
OK, COPY JAKE>CARD>ENGINE.PROB CARS>COMPLAINTS>JAKE.ENGINE
```

3. Press Ctrl D to delete the D.

```
OK, COPY JAKE>CAR>ENGINE.PROB CARS>COMPLAINTS>JAKE.ENGINE
```

4. Enter S.

```
OK, COPY JAKE>CARS>ENGINE.PROB CARS>COMPLAINTS>JAKE.ENGINE
```

5. Press Return.

ECL remembers the last 200 command lines executed during the current login session. This list of commands is called the **command history**. Pressing Ctrl Z in succession lets you individually review the command history. The section, Command History, describes other ways to use the list of commands that ECL remembers.

Searching for Previously Entered Text

You can redisplay a command line from the command history by specifying a unique search string. ECL has two commands, Esc S (**forward_search**) and Esc R or Ctrl R (**reverse_search**), that prompt for a character string. Esc R searches backward through your command lines for the string entered, beginning with the most recently entered command line. Esc S begins the search with the oldest command line in the command history.

After you enter the string, ECL performs a case-insensitive search in the specified direction for the most recently executed command line containing that string. ECL searches the last 200 PRIMOS command lines entered until it finds the string specified. If the string is not there, the terminal bell beeps.

ECL maintains a search buffer containing the 10 most recently requested search strings. If you press Return at the search prompt instead of entering a string, ECL looks for the most recently entered search string.

The following example shows how to use the search command to find a previously entered pathname. When you press Esc R on a command line, ECL displays its search prompt, shown below. Enter the string that you want to find. For example,

```
R-search: DINOS
```

ECL displays the last command line entered that contains the string DINOS.

```
OK, EMACS GIFTS>FRIENDS>DINOS
```

You can edit this line as usual. To execute the command line, press Return.

Table 7-3 shows the ECL commands for searching through command-line entries.

TABLE 7-3
Search Commands

ECL Command	*Direction*	*Command Name*
Esc R or Ctrl R	Backward	**reverse_search**
Esc S	Forward	**forward_search**

Aborting ECL Commands

Enter Ctrl G to abort an ECL operation in progress and display the last command line in the command history. This is especially useful if you are searching for or displaying previous command lines and want to return to the end of the command history.

Advanced ECL Commands

This section describes more advanced functions that you can perform with ECL:

- Repeating ECL commands
- Moving and deleting words
- Changing case and character position
- Moving and copying regions of text
- Restoring copied and deleted text
- Using the command history
- Expanding pathnames
- Referencing other directories
- Expanding abbreviations
- Using command macros
- Customizing your ECL prompts
- Using the password command
- Using extended commands
- Using ECL across a network

Repeating ECL Commands

ECL lets you specify a number of times to repeat the next ECL command you enter. Use the **esc_digit** repeat command:

1. Press Esc and the digit for the number of times you want the command executed. For example, Esc 3.
2. Enter the ECL command you want repeated.

You must specify the number of times *before* entering the ECL command to be repeated. For example, if the cursor is in the middle of a command line and you want to delete the two characters to the right of the cursor, press Esc 2 Ctrl D. To delete 12 characters to the right of the cursor (including spaces), press Esc 1 2 Ctrl D.

The **multiplier** command, Ctrl U, works similarly to the **esc_digit** repeat command. To repeat an ECL command 12 times you can use either Ctrl U 1 2 or Esc 1 2. Unlike **esc_digit**, however, Ctrl U used alone repeats the next command four times. For instance, pressing Ctrl U Ctrl D deletes the next four characters on the command line. When preceded with a repeat count, Ctrl U multiplies that count by four. For example, if you press

Ctrl U Ctrl U Ctrl F

ECL moves the cursor forward by 16 (4 x 4) characters. You can also use the **esc_digit** and **multiplier** commands together. For instance,

Esc 1 2 Ctrl U Ctrl F

moves the cursor ahead 48 (12 x 4) characters. If you use these two commands together, you must enter the repeat command first.

You can also repeat the *previous* ECL command with Ctrl C. To perform the previous ECL command more than once, enter the **esc_digit** or **multiplier** command followed by Ctrl C. If the previous command also had a repeat count associated with it, ECL multiplies the two counts and performs the previous command that number of times.

ECL never considers Ctrl C as the previous command. Therefore, if you enter Ctrl C twice, ECL reexecutes the command entered before the first Ctrl C command.

Table 7-4 summarizes the commands to repeat ECL commands.

Moving and Deleting Words

ECL defines a word as a string of alphanumeric and underscore characters only; spaces and any other characters, such as punctuation, are word delimiters. Therefore, ECL considers the string

```
DATE; SLIST SQUID_RECIPE
```

to be three words long, while

```
DATE; SLIST SQUID.RECIPE
```

is four words long, because it contains the delimiter (.).

TABLE 7-4
Repeat and Multiplier Commands

ECL Command	*Action*	*Command Name*
Esc *n* or Ctrl U *n*	Repeats next ECL command *n* times	**esc_digit**
Ctrl U	Repeats next ECL command four times	**multiplier**
Esc *n* Ctrl U	Repeats next ECL command a multiple *n* of four times	**multiplier**
Ctrl U Ctrl U	Repeats next ECL command 16 times	
Ctrl C	Repeats previous ECL command	**reexecute**

You can instruct ECL to consider all legal PRIMOS filename characters as valid characters rather than as word delimiters with the ECL –ENTRY option. Refer to the section, ECL Command Options, at the end of this description.

Table 7-5 lists the commands for cursor movement by words and for deletion of words.

TABLE 7-5
Moving the Cursor and Deleting by Words

ECL Command	*Action*	*Command Name*
Esc B	Moves backward one word	**back_word**
Esc F	Moves forward one word	**forward_word**
Esc D	Deletes next word	**delete_word**
Esc Ctrl H or Esc Backspace	Deletes previous word	**rubout_word**

Changing Case and Character Position

ECL permits some basic character modifications:

- Changing characters and words to uppercase
- Changing characters and words to lowercase
- Transposing two characters to the left of the cursor

ECL defines words as described in the previous section, Moving and Deleting Words.

Table 7-6 shows the ECL commands that perform uppercase and lowercase conversion, as well as character transposition. The case change occurs from the character where the cursor is positioned to the end of the word.

TABLE 7-6
Case and Character Position Commands

ECL Command	*Action*	*Command Name*
Ctrl ^	Changes lowercase character to uppercase or vice versa	**toggle_case**
Esc U	Changes word to all uppercase	**upcase_word**
Esc L	Changes word to all lowercase	**dncase_word**
Ctrl T	Transposes two characters to left of cursor	**twiddle**

Defining, Copying, and Deleting Regions of Text

With ECL you can define a region of text that you can copy, delete, and insert into one or more command lines. A **region** is a block of text that you define to ECL by specifying its boundaries. This section describes defining, copying, and deleting regions. The next section describes how to restore copied and deleted regions.

Defining Regions of Text: Use Ctrl @ to define one of the region's boundaries, called the **mark**. To define the other boundary, move the cursor to the place where you want the region to end. The region is the area of text between the mark and the new cursor position. Within ECL, the maximum size for a region is approximately one PRIMOS command line (a maximum of 158 characters). By default, the mark is located at the first position to the right of the prompt.

The mark does not appear on your terminal screen; your command line looks identical regardless of whether or not you have defined a mark position. To check a region's current boundaries, use Ctrl X Ctrl X to swap the mark and cursor positions. Ctrl X Ctrl X does not change the boundaries of the region.

Copying Regions of Text: Escape W copies a defined region and stores it in a buffer for subsequent retrieval. Esc W does not change your command line; the region that you define and copy is still displayed. However, you can redisplay and reexecute the commands in the copied region by using the yank command, explained in the next section, Restoring Copied and Deleted Text.

Deleting Regions: Use Ctrl W to delete a defined region. If you use Ctrl W with a repeat of 2 or more (such as Esc 2), the command deletes the entire line.

Table 7-7 summarizes the ECL commands to define regions, check their boundaries, and copy and delete them.

TABLE 7-7
Region Commands

ECL Command	*Action*	*Command Name*
Ctrl @	Defines one end of a region	**mark**
Ctrl X Ctrl X	Checks region boundaries	**exchange_mark**
Esc W	Copies a region into a buffer	**copy_region**
Ctrl W	Deletes a region	**kill_region**

Restoring Copied and Deleted Text

ECL maintains a 10-item buffer containing the most recent

- Word deletions
- Line deletions
- Region deletions
- Copied regions

You can redisplay the most recently deleted or copied text in this buffer with the ECL **yank** command, Ctrl Y. Ctrl Y inserts the most recent buffer entry to the left of the cursor. If you press Ctrl Y several times in succession, the same entry (the last region deleted or copied) is displayed. Specifying the **esc_digit** command (Esc *n*) before Ctrl Y redisplays the *n*th most recently deleted or copied text region.

Note

You cannot restore character deletions performed with the PRIMOS erase character or with Ctrl D because ECL does not store these deletions in a buffer.

Ctrl Y displays the most recent buffer entry. To replace it with the second most recent and previous entries, use Esc Y. Unlike Ctrl Y, each time you press Esc Y, ECL changes the pointer within the buffer to the previous entry and then displays that entry, replacing the previous entry if you haven't moved the cursor. With Esc Y you can view from the second most recent to the tenth most recent buffer entries. If you continue pressing Esc Y after reaching the last item in the buffer, you review the same entries again in the same order, because the buffer is circular.

Table 7-8 summarizes the two ECL commands that redisplay deleted or copied text.

The Command History

In addition to a buffer of deleted and copied command line text, ECL also maintains a list of command lines executed. Every PRIMOS command line that you execute becomes part of your ECL command history. The **command history** is a list of a maximum of 200 of the

TABLE 7-8
Commands to Redisplay Deleted and Copied Text

ECL Command	*Action*	*Command Name*
Ctrl Y	Redisplays most recent buffer entry	**yank**
Esc Y	Redisplays from second to tenth most recent buffer entry	**yank_replace**

most recently executed command lines. You can save and restore command histories. You can also use them to display

- All command lines in the list
- A specified number of the most recently executed command lines
- A command line meeting specified search criteria
- A command line of the indicated line number
- The previous command line
- The next command line (if you are not positioned at the end of the command history)

Displaying the Command History: To display the current command history, use the **esc_digit repeat** and **refresh** commands. Press Esc 2 0 0 Ctrl L. Specifying a repeat of 200 ensures that you see a maximum of 200 of the most recently entered PRIMOS command lines. If your current history is smaller than that, ECL does not display the unused entries.

ECL automatically numbers each command line in the display. A sample command history is shown below.

```
001) DATE
002) A *>NEW.PRODUCTS; LD
003) EMACS TENNIS.BALL.RETRIEVER
```

The command history does not contain deletions; ECL maintains a separate buffer for them. As with the buffer for deletions and copied regions, the command history is circular. ECL overwrites the oldest (least recent) commands when you have entered more than 200 commands as part of the same command history. For example, command number 201 becomes command 001, 202 becomes 002, and so on. Your command history contains the following sequence:

```
199)
200)
001)
002)
```

Moving Around in the Command History: Several ECL commands let you redisplay commands within the command history. Ctrl Z, described earlier, lets you move back through the history one command line at a time. Another command, Ctrl N, lets you move forward through the history again. You can also redisplay a specific command line with the **goto_line** command, Esc *line-number* Esc G. For example, pressing

Esc 2 Esc G

redisplays the second command in the history:

```
002) A *>NEW.PRODUCTS; LD
```

Note

In the command history examples, the command history line numbers are shown with each command line. By default, command history line numbers only appear with command lines when you display the command history with Escape *n* Ctrl L. However, the section, Customizing Your ECL Prompts, shows how to make the command history line number appear as part of your prompt in order to create a display like the one in the examples.

To move from this position to the end of the history, you can use Ctrl N repeatedly until you reach it. An easier way to get to the end of the history is to use Ctrl G (**abort_cmd**) or the **goto_line** command with a line number of zero (Esc 0 Esc G).

Table 7-9 lists ECL commands for displaying specific command lines from the command history, as well as commands for displaying the entire list.

TABLE 7-9
Command Line Display Commands

ECL Command	*Command Line Display*	*Command Name*
Esc 2 0 0 Ctrl L	All (entire list)	**refresh**
Escape *n* Ctrl L	Previous number of lines	**refresh**
Ctrl L	Current (redisplay)	**refresh**
Esc *line-number* Esc G	Line number specified	**goto_line**
Ctrl N	Next line	**next_line**
Ctrl Z	Previous line	**prev_line**

Staying Positioned Within the Command History: An internal **pointer** indicates your position within the command history. By default, after executing a previously entered command, the pointer is positioned at the end of the command history.

In the partial command history below, the pointer is at line 27, which is currently empty.

```
023) EMACS RECENT.SIGHTINGS
024) DATE
025) EMACS IFO.LOG
026) SPOOL IFO.LOG
```

To edit the RECENT.SIGHTINGS file again, issue the previous line command four times (Ctrl U Ctrl Z or Esc 4 Ctrl Z) to redisplay and reexecute line 23. After finishing the editing session, the command line below is displayed.

```
027) OK,
```

If you have a series of command lines that you want to reexecute, it is more convenient if the pointer moves with you in the history, instead of going to the end of the list. The ECL –STICK option lets you do this. Specify the –STICK option at any point in a terminal session by typing

```
OK, ECL -STICK
```

Using the above example, if you specify the –STICK option and reexecute line 23, the next line displayed is

```
024) OK,
```

The Hidden Command: You must be careful when using the –STICK option because it does not let you see text on the current command line. For instance in the above example, line 24 actually contains the PRIMOS DATE command, although ECL does not automatically display it. A command line containing text that you cannot see is called a **hidden command line**. If you enter something else on a hidden command line, ECL overwrites the invisible, previously entered text with your new entry.

To prevent inadvertent overwriting of command lines, use the ECL –SHOW_HIDDEN option with the –STICK option. With the –SHOW_HIDDEN option, ECL displays the next command line, complete with text. To specify the –SHOW_HIDDEN option, type

```
OK, ECL -SHOW_HIDDEN
```

Hidden commands occur only when you use the –STICK option; therefore, the –SHOW_HIDDEN option is useful only if you have also specified the –STICK option.

If the –SHOW_HIDDEN option is invoked in the previous example, line 24 looks like this:

```
024) OK, DATE
```

After executing line 24, the command line below appears.

```
025) OK, EMACS IFO.LOG
```

Note that when a hidden command is revealed in this way, the cursor appears at the beginning of the command line instead of at the end, where it appears for all other commands. A revealed hidden command stays on the command line only if you issue an ECL editing command, such as Ctrl D to delete a character. If instead you begin entering text, the revealed command automatically disappears to prevent the hidden command from interfering with the new text.

By specifying both the –SHOW_HIDDEN and –STICK options, you can sequentially redisplay and reexecute a series of commands. The –NO_SHOW_HIDDEN and –NO_STICK options reset the default state, so that the pointer automatically moves to the end of the command history after each command.

Saving the Original Version of Edited Commands: If you edit and reexecute a previously entered command, the edited version replaces the original version in the command history. To prevent this, use the ECL –STACK option. Specify the –STACK option at any point during a terminal session by typing

```
OK, ECL -STACK
```

If you specify the –STACK option, ECL adds edited command lines to the end of the command history after you execute them, instead of overwriting the original command lines. For example, suppose you invoke the –STACK option on line 27 of the previous example. If you then return to line 23 and change the filename to OLD.SIGHTINGS, the command history after your EMACS session looks like this:

```
023) EMACS RECENT.SIGHTINGS
024) DATE
025) EMACS IFO.LOG
026) SPOOL IFO.LOG
027) ECL -STACK
028) EMACS OLD.SIGHTINGS
```

Saving and Restoring the Command History: If you have a series of commonly used command lines, you can store them in a command history file. To file a command history, use the –SAVE_HISTORY option after executing the commands you want to store. The format is

ECL –SAVE_HISTORY *filename*

The *filename* entered contains a maximum of the last 200 command lines executed, as well as the 10 most recent deletions and copied regions, and the 10 most recently entered search strings.

Note

Do not enter the ECL –SAVE_HISTORY command at the beginning of a login session. The only commands saved in the file specified are those that you enter prior to the ECL –SAVE_HISTORY command; commands entered after the ECL –SAVE_HISTORY command are not placed in the file.

To reactivate a command history file, use the –RESTORE_HISTORY option. The format is

ECL –RESTORE_HISTORY ***filename***

The above command makes the command history stored in *filename* your current command history. The commands in the file replace any active command history you may have. If you restore a command history file as soon as you log in, new command lines are appended to the restored command history, and become part of your current list.

Note

You cannot display command history files with the PRIMOS SLIST command, nor can you print them. To restore and reexecute the command lines in files, you must use the ECL commands and options discussed in this chapter. To create a regular PRIMOS file containing the command history, create a command output file and specify the ECL –NO_CLEAN_COMO option. Typing Esc 2 0 0 Ctrl L displays the command history and writes it to the command output file.

Expanding Pathnames

ECL can perform automatic pathname completion for files and directories on your system. When you use the ECL **expand_wild** command by pressing Ctrl I or TAB, ECL tries to complete what you've typed with a filename or directory name that matches your entry. The **expand_wild** command works similarly to the PRIMOS LD command with a partial pathname, followed by wildcard characters (@@).

When you use the **expand_wild** command, ECL does one of the following:

- If two or more file system objects match the partial pathname entered, ECL completes what you've typed with the portion of the pathname that is common to all of them. Press Ctrl I again to see the list of all files and directories containing the partial pathname.
- If only one file system object matches the partial pathname entered, ECL replaces the string entered with the name of that object. If this is a file, ECL redisplays the filename when you press Ctrl I a second time. If the matching pathname expands to a directory name, ECL automatically adds a greater-than sign (>) to the end of it. In this case when you press Ctrl I a second time, ECL tries to expand within this directory, which usually results in the display of the directory's contents.
- If no file system objects match the partial pathname entered, the terminal bell beeps and the command line remains the same.

ECL displays possible command-line completions in a numbered menu. In the menu, directory names end with a greater-than sign (>).

Note

ECL does not support expansion of pathnames with passwords or passworded directories.

Here's an example of how the **expand_wild** command works. In the command lines below, the underscore character (_) represents the cursor.

```
OK, SLIST REV_
Ctrl I
```

The previous command line changes to

```
OK, SLIST REVIEW._
```
Ctrl I

The following menu is displayed:

```
   1) REVIEW.1    2) REVIEW.2   3) REVIEW.DRAFT    4) REVIEW.OLD>

OK, SLIST REVIEW._
```

Notice in the above example that ECL completes pathnames that start with the characters entered (this is the default), and produces a final listing similar to the display shown by

```
OK, LD REV@@
```

Choosing a Pathname Completion: After you have expanded a pathname, you can either type the rest of the command line yourself or choose one completion from the menu with the **expand_wild_menu** command. You use this command by selecting a number from the menu and typing

Esc *n* Esc Ctrl I

where *n* is the number you have selected. ECL then completes the pathname in your command line with the completion you have selected.

For example, you can choose the completion `REVIEW.DRAFT` from the above menu by typing

Esc 3 Esc Ctrl I

ECL then completes the pathname as

```
OK, SLIST REVIEW.DRAFT
```

Entry number 4 in the display ends with a greater-than character (>) indicating that REVIEW.OLD is a directory. The > character helps you complete the pathname by adding further elements, such as filenames. Sometimes, however, you want to use the directory name itself in a command such as ATTACH. In this case, you must remove the > character, since PRIMOS does not accept pathnames that end with >. You can prevent ECL from adding the > character by using the –NO_WILD_DIRECTORY option. Type

```
OK, ECL -NO_WILD_DIRECTORY
```

at any point in the terminal session.

The ECL –WILD_TAIL option, in conjunction with the **expand_wild** command, lets you complete pathnames that end with a particular character string. Its use is similar to entering the PRIMOS LD command with wildcards placed before a string of pathname characters (such as LD @@CPL) or in the middle of a string (such as LD REV@.RUN). To use the

–WILD_TAIL option, position the cursor exactly where you want the wildcarding to occur and then press Ctrl I. The **expand_wild** command with the –WILD_TAIL option in force produces the same display format as shown previously.

The following example illustrates the use of the –WILD_TAIL option.

```
OK, LD <DISK1>USER1>DIRECTORY1>TXT
Ctrl I
```

ECL then displays the files within DIRECTORY1 that end in TXT, shown below.

```
  1) FILE1.TXT   2) FILE2.TXT
```

You can use both the **repeat** and **multiplier** commands with either version of the **expand_wild** command; doing so skips the intermediate step of partial expansion. The previous example is altered to demonstrate this.

```
OK, REV_
```

After you press Esc 2 or Ctrl U 2 and then press Ctrl I, you see the following display:

```
 1) REVIEW.1   2) REVIEW.2   3) REVIEW.DRAFT   4) REVIEW.OLD>

OK, REV_
```

Note that the final result is different only in that the PRIMOS command line at the bottom is the same as the one originally entered, not the partially expanded version produced by pressing Ctrl I alone (without preceding it with the **repeat** or **multiplier** commands).

Referencing Other Directories

You can use the **expand_wild** command in conjunction with ECL commands to refer to directories other than the current directory. These referencing commands enable pathname completion in directories above and including your current directory; these commands do not change your current attach point.

The format of displays produced by referencing different directories is very similar to those produced for pathname completion within the current directory. The example below shows how to refer to a parent directory.

```
OK, EMACS *<P_
Ctrl I
```

The following display appears.

```
  1) PALS            2) PICNIC           3) PLANES>
  4) PLANS.BIG>      5) PLANS.SMALL>     6) POINTS
  7) PROMISES>       8) PROPERTY>        9) PROSPECTS

OK, EMACS *<P_
```

A very similar display appears if you attach to the directory above the current directory and enter

```
OK, LD P@@
```

The difference is that with Ctrl I, you don't have to attach to the directory or enter its pathname to view its contents. Also, ECL redisplays the original command line at the bottom of the listing (in this case, EMACS *<P), so that you can complete the rest of the filename that you want to edit.

In the previous example, ECL listed all file system objects beginning with the letter P that you could view from the parent directory. However, if you enter

```
OK, EMACS *<PI_
Ctrl I
```

ECL changes the command line to

```
OK, EMACS *<PICNIC_
```

because the PICNIC file is the only possible expansion for a string of PI. If you enter

```
OK, EMACS *<PE_
Ctrl I
```

the terminal beeps, because the parent directory does not contain any file system objects that begin with those letters.

To display the contents of a higher-level directory, enter *< Ctrl I for the parent directory or *<< Ctrl I for the grandparent directory. You can refer to even higher-level directories by including additional less-than signs to the right of the asterisk.

Table 7-10 lists the referencing commands.

TABLE 7-10
Reference Commands

Reference Command	*Reference Position*
*<[*string*] Ctrl I	Parent directory
*<<[*string*] Ctrl I	Grandparent directory
*> Ctrl I	Current directory
<*DISK*>*directory*>[*string*] Ctrl I or *directory*>[string] Ctrl I	Absolute directory

If you reference a directory without entering a partial pathname for completion within that directory, such as a pathname ending with >, this usually results in display of that directory's contents.

Finding What a Key Does

You can find out what command is bound to any key sequence by typing Ctrl _, the ECL **explain_key** command. When you type Ctrl _, ECL prompts you for the key combination you want explained. You then type the key combination, and ECL tells you which ECL command it is bound to.

Expanding Abbreviations

To expand abbreviations in a command line, move the cursor to the abbreviation you want to expand and type Esc A. ECL replaces the abbreviation with its expansion. You can use this to find out exactly what an abbreviation does. It also makes it easy to use an abbreviation in a slightly modified form. Just type the abbreviation in your command line, expand it with Esc A, and modify it.

If your abbreviation contains global variables, they are also expanded to reflect their current values.

If a command line contains more than one abbreviation, you can have ECL expand them all by preceding the **expand_abbrev** command with Esc 4. In other words, you type

```
Esc 4 Escape A
```

To expand all abbreviations and global variables, type

```
Esc 1 6 Escape A
```

If the command line begins with the PRIMOS syntax suppression character ~, abbreviations are not expanded.

Note

Some commands, like **expand_abbrev**, don't repeat when you precede them with Esc *n*. Instead, the number that you type, called a **numeric argument**, causes the command to behave in a different fashion. For example, the numeric argument 4 causes **expand_abbrev** to expand all the abbreviations in a command line instead of just expanding the one the cursor is on.

Keyboard Macros

You may find yourself repeating a sequence of ECL commands several times during a terminal session. You can avoid having to retype the commands each time by creating a keyboard macro. A **keyboard macro** is a collection of keystrokes that ECL stores and can replay on command.

Collecting a Macro: Before you can use a macro, ECL must record it. This is called **collecting** a macro. You begin collecting a macro by issuing the ECL **collect_macro**

command, Esc (. After you type Esc (, ECL records subsequent keystrokes. The macro can include any set of keystrokes, including ECL commands and text. Return ends a macro, so a macro cannot be longer than a single command line.

While ECL records your keystrokes, it also processes them normally. Any ECL commands you give are carried out, and any text you type is displayed on the screen.

When you have finished collecting the macro, either type Return or issue the ECL **finish_macro** command, Esc). This tells ECL to stop recording your keystrokes in the macro.

You can record up to about 200 keystrokes in a macro. You can only have one macro at a time. If you collect a new macro, it replaces a previous one. However, *while* you are collecting a macro, the most recently defined previous macro is still available. The next section shows you how you can make use of this feature. You cannot include the **collect_macro** command (Esc () in a macro.

Executing a Macro: Once you have collected a macro, you can replay it as often as you want during the same terminal session using the ECL **execute_macro** command, Esc E. When you press Esc E, ECL reproduces all of the keystrokes contained in the macro, just as if you were typing them at the keyboard at that point. All the ECL commands included in the macro are executed, and any text included in the macro is reproduced exactly as if you were typing it at the terminal.

You can give a repeat argument to the **execute_macro** command just as you would to other ECL commands:

Esc *n* Esc E

where *n* is the number of repetitions. However, only the first ECL command or text character contained in the macro is repeated.

When you are collecting a new macro, you can include the most recently defined previous macro by giving the **execute_macro** command.

To find out the contents of a macro, use the ECL **explain_key** command. Type Ctrl _. When ECL prompts you for a key sequence, type Esc E, and ECL displays the contents of the current macro.

Table 7-11 lists the macro related commands.

TABLE 7-11
Macro Commands

Command	*Command Name*
Escape E	**execute_macro**
Escape (	**collect_macro**
Escape)	**finish_macro**

A Macro Example: The following example shows how to use a macro to modify and reexecute a series of previous command lines. (The example assumes that the –STACK option has not been invoked.) Suppose that during a terminal session you give a series of PRIMOS COPY commands with relative pathnames (that is pathnames beginning with *>).

```
025) OK, COPY *>VEGETABLE>TOMATO  MENU>FIRST
                .
                .
035) OK, COPY *>FRUIT>APPLE MENU>SECOND
036) OK, COPY *>NUT>CASHEW  MENU>THIRD
                .
                .
041) OK, COPY *>HERB>THYME MENU>FOURTH
                .
                .
075) OK,
```

Later in the session, you are working in another directory and want to repeat the COPY commands. You can use a macro to replace the *> with the directory name in each of the command lines.

First type Esc (to begin recording the macro. Then issue the necessary commands to search for and edit one of the earlier command lines. For example, type Esc R, and when ECL prompts you for the search text, type

```
        R-Search: COPY *>
```

This causes ECL to search for and redisplay line 41:

```
041) OK, COPY *>HERB>THYME MENU>FOURTH_
```

Now edit the line by moving the cursor to the beginning of the line(Ctrl A), then forward six characters (Esc 6 Ctrl F), deleting the * (Ctrl D), and typing in the new directory name, PLANTS. The new version of line 41 now looks like this:

```
041) OK, COPY PLANTS>HERB>THYME MENU>FOURTH
```

Type Return to execute the new command line and stop recording the macro.

The entire sequence of key strokes used to edit and execute line 41 is now saved as a macro. You can use the same sequence to find, edit, and execute all of the previous lines beginning with the same text (COPY *>).

Type Esc E to execute the macro. ECL finds, edits, and executes line 36, displaying the edited version of the line:

```
036) OK, COPY PLANTS>NUT>CASHEW MENU>THIRD
075) OK,
```

Type Esc E again, and ECL carries out the same sequence of operations on line 35, displaying

```
035) OK, COPY PLANTS>FRUIT>APPLE MENU>SECOND
075) OK,
```

You can repeat this operation until you have found, edited, and executed all of the command lines.

Customizing Your ECL Prompts

By default, ECL displays the PRIMOS prompts that are in effect when you invoke ECL. If you are using the default PRIMOS prompts, you see the default prompts as you work with ECL, as well. If you customize your prompts with the PRIMOS RDY command, ECL also displays the new prompts. If you change your PRIMOS prompt by giving the PRIMOS RDY command while you are using ECL, ECL displays the new prompt.

ECL can also display its own prompts. Set the ECL ready, error, and warning prompts with the ECL –READY_BRIEF, –ERROR_BRIEF, and –WARNING_BRIEF options. You can also include ECL line numbers in your prompt by including a pound sign (#) in the prompt text. As when you customize PRIMOS prompts, put the entire string in single quotation marks. For example,

```
OK, ECL -READY_BRIEF '#) YOU RANG? '
```

sets your ECL ready prompt to

```
001) YOU RANG?
```

ECL prompts that you set supersede any prompts you set with the PRIMOS RDY command. Once you have specified ECL prompts, they remain in effect until you invoke the ECL –INITIALIZE option or quit ECL.

WARNING

The –INITIALIZE option resets all ECL options to their default values and clears the command history, search, copy, and delete buffers.

The Password Command

Sometimes you type command lines that contain passwords or other sensitive material that you may not want echoed to the screen or saved in the command history. Use the ECL password command in these circumstances: Esc P. You must give the password command at the beginning of the command line, before you type anything else. Whatever you type on the current command line is then sent to the PRIMOS command processor without being echoed to the screen or saved in the command history.

The ECL editing commands are disabled while you type the invisible command line. You can use only the PRIMOS erase and kill characters to edit the line.

Extended Commands

You normally invoke each of the ECL commands introduced in this chapter by typing a key sequence made up of the Esc, Ctrl, and other keys in various combinations. For example, to move the cursor forward one character, you type Ctrl F. The key sequence you type to execute a command is said to be **bound** to that command.

Each of the commands introduced here also has a **command name**. For example, the command to move forward one character is called **forward_char**. The key sequence Ctrl F is thus bound to the command **forward_char**. You can also execute an ECL command by using its command name instead of the key sequence bound to it. You do this with the ECL **extend_command,** Esc X.

When you type Esc X, ECL prompts you for the name of the command you want to execute. When you type the command name, ECL executes the command just as if you had given the key sequence bound to that command.

Using ECL Across a Network

If you use ECL on a remote system, specify the –MODE REMOTE_ECHO option when you establish the connection via NETLINK. This speeds up the terminal display. You must open any command output (COMO) files after you are on the remote system for the ECL –CLEAN_COMO option (the default) to work. If you open a COMO file on the local system and then attach to a remote system, the file contains all ECL dialog.

ECL Command Summary

Table 7-12 includes all of the ECL commands discussed in this chapter, grouped according to function. For further information about ECL, and a complete listing of ECL commands, see the *PRIMOS Commands Reference Guide.*

An asterisk (*) indicates functions that accept the **repeat** (Esc *n*) and **multiplier** (Ctrl U) commands.

TABLE 7-12
ECL Commands by Function

Command Sequence	*Command Name (precede with* Esc X*)*	*Action*
Case and Character Position Commands		
Ctrl ^	**toggle_case**	Changes lowercase character to uppercase and vice versa*
Esc L	**downcase_word**	Changes word to all lowercase*
Ctrl T	**twiddle**	Transposes two characters to left of cursor
Esc U	**upcase_word**	Changes word to all uppercase*
Command Line Display Commands		
Escape 2 0 0 Ctrl L		Displays command history
Escape *line-number* Esc G	**goto_line**	Displays line number specified
Ctrl L	**refresh**	Redisplays current line
Esc *n*		Redisplays previous *n* lines Ctrl L
Esc 0 Ctrl N		Explicitly displays hidden command
Ctrl N	**next_line**	Displays next line*
Ctrl Z	**prev_line**	Displays previous line*
Copied and Deleted Text Redisplay Commands		
Ctrl Y	**yank**	Redisplays most recent buffer entry*
Escape Y	**yank_replace**	Redisplays from the second to the tenth most recent buffer entries*

TABLE 7-12 - Continued
ECL Commands by Function

Command Sequence	*Command Name (precede with* Esc X*)*	*Action*
Cursor-moving Commands		
Ctrl A	**begin_line**	Moves to beginning of line
Ctrl B	**back_char**	Moves back one character*
Escape B	**back_word**	Moves back one word*
Ctrl E	**end_line**	Moves to end of line
Ctrl F	**forward_char**	Moves forward one character*
Esc F	**forward_word**	Moves forward one word*
Delete Commands		
Ctrl D	**delete_char**	Deletes next character*
Esc D	**delete_word**	Deletes next word*
Ctrl H or Backspace	**rubout_char**	Deletes previous character*
Esc Ctrl H or Esc Backspace	**rubout_word**	Deletes previous word*
Ctrl K	**kill_line**	Deletes rest of line (from cursor to end)
Ctrl U Ctrl W		Deletes entire line
Ctrl W	**kill_region**	Deletes region specified with mark command*
Macro Commands		
Esc E	**execute_macro**	Executes most recently collected macro
Esc (	**collect_macro**	Starts recording command macro
Esc)	**finish_macro**	Stops recording command macro

TABLE 7-12 - Continued
ECL Commands by Function

Command Sequence	*Command Name (precede with* Esc X*)*	*Action*
Miscellaneous Commands		
Ctrl G	**abort_cmd**	Aborts the most recent ECL command
Esc P	**password**	Prevents screen echo of line
Esc X	**extend_command**	Executes named command
Esc A	**expand_abbrev**	Expands abbreviations in command line
Pathname Expansion Commands		
Ctrl I	**expand_wild**	Completes rest of common pathname
Esc *entry-number* Esc Ctrl I	**expand_wild_menu**	Picks entry from wild display menu
*<[*string*] Ctrl I		Refers to parent directory
*<<[*string*] Ctrl I		Refers to grandparent directory
<DISK>directory>[*string*] Ctrl I or *directory>*[*string*] Ctrl I		Refers to absolute directory
Region Commands		
Ctrl X Ctrl X	**exchange_mark**	Checks region boundaries
Esc W	**copy_region**	Copies region into a buffer
Ctrl @	**mark**	Defines one end of a region

TABLE 7-12 - Continued
ECL Commands by Function

Command Sequence	*Command Name (precede with* Esc X*)*	*Action*
Repeat and Multiplier Commands		
Ctrl C	**reexecute**	Repeats previous ECL command*
Esc *n* or Ctrl U *n*	**esc_digit** **multiplier**	Repeats next ECL command *n* times
Ctrl U	**multiplier**	Repeats next ECL command four times
Esc *n* Ctrl U		Repeats next ECL command a multiple *n* of four times
Ctrl U Ctrl U		Repeats next ECL command 16 times
Search Commands		
Esc R	**reverse_search**	Searches backward*
Esc S	**forward_search**	Searches forward*

ECL Command Options

This section lists the most common options that you can issue with the PRIMOS ECL command. The list is arranged alphabetically. Default options and their counterparts are paired, with the default appearing first. When an option has no default, you must explicitly give the option with the ECL command if you want the option to take effect. Valid abbreviations are shown in red. For a complete list of ECL options, see the *PRIMOS Commands Reference Guide*.

Option	*Meaning*
–CLEAN_COMO –NO_CLEAN_COMO	Controls whether ECL terminal output appears in a command output file that you open with the PRIMOS COMO command. With the default, –CLEAN_COMO, only the ECL prompt and the final version of each command line (when you press Return) appears in the command output file. –NO_CLEAN_COMO causes all screen output to be included in the file. If you work across a network and use a command output file, you must open the file on the remote system for the –CLEAN_COMO option to work properly.

–COMPONENT **–ENTRY**	Defines characters that ECL considers valid for words. The default, –COMPONENT, indicates that words can consist only of alphanumeric and underscore characters. –ENTRY adds the characters # $ & * / - . @ + ^ = (those valid for PRIMOS file system objects). The definition of word affects ECL commands such as Esc F (**forward_word**), Esc B (**back_word**), and Esc D (**delete_word**).
–ERROR_BRIEF ***prompt***	Sets the error prompt displayed within ECL to *prompt*. The section, Customizing Your ECL Prompts, shows an example of how to reset your prompts.
–HELP	Displays the valid ECL options.
–INITIALIZE	Reinitializes ECL to its default options and clears the command history as well as the search, copy, and delete buffers.
–NO_CASE_SEARCH **–CASE_SEARCH**	Controls whether searches for strings entered are case-sensitive. With the default –NO_CASE_SEARCH, searches are not case-sensitive. –CASE_SEARCH makes searches case-sensitive.
–NO_EDIT_COMI **–EDIT_COMI**	Controls whether ECL processes the contents of command input files. With the default –NO_EDIT_COMI, the contents of command input files are passed directly to PRIMOS for execution without processing by ECL. Command lines are not included in the command history. –EDIT_COMI causes ECL to treat input from command input files exactly like terminal input and to include commands in the command history.
–NO_SHOW_HIDDEN **–SHOW_HIDDEN**	Controls whether or not ECL displays hidden commands. With the default –NO_SHOW_HIDDEN, hidden commands are not displayed. –SHOW_HIDDEN causes ECL to display hidden commands.
–NO_STACK **–STACK**	Controls whether ECL considers the command history to be a ring (–NO_STACK) or a stack (–STACK). With a ring, modifications to command lines already executed replace the original commands in the command history. With a stack, each modification becomes a new command in the command history. Stacks minimize lost commands, but make executing a sequence of previous commands more difficult.
–NO_STICK **–STICK**	With the default –NO_STICK, the command history pointer is positioned at the end of the command history after you redisplay and execute a previous command. The –STICK option causes ECL to position the command history pointer at the command immediately after any redisplayed command that you execute. With –STICK, you can execute a sequence of previous commands. Refer to the section, Changing the Command History Pointer, for more information.
–NO_WILD_TAIL **–WILD_TAIL**	Controls whether automatic pathname completion performs as if wildcards are appended at the end of a string –NO_WILD_TAIL, the default; or inserted at the cursor position (–WILD_TAIL).

–OBEY_ERKL **–NO_OBEY_ERKL**	Controls whether or not ECL observes the characters defined as your PRIMOS erase and kill characters (using the PRIMOS TERM command). With the default –OBEY_ERKL, ECL uses whatever erase and kill characters are defined to PRIMOS. –NO_OBEY_ERKL allows these characters to perform other functions in ECL.
–OFF	Turns off the ECL environment and reverts to PRIMOS command line processing. ECL continues to store all option settings while you are logged in. If you give the ECL –ON command during the same terminal session, the stored options remain in effect.
–ON	Invokes the ECL editor.
–READY_BRIEF ***prompt***	Sets the ready prompt within ECL to *prompt*. Refer to the section, Customizing Your ECL Prompts, for an example of how to do this.
–RESTORE_HISTORY ***filename***	Makes the command history contained in *filename* the current command history. Refer to the section, Saving and Restoring the Command History, for more information.
–ROW_MAJOR **–COL_MAJOR**	Indicates whether the **expand_wild** command sorts alphabetical lists horizontally (–ROW_MAJOR, the default) or vertically (–COL_MAJOR).
–SAVE_HISTORY ***filename***	Stores the current command history in *filename*. Refer to the section, Saving and Restoring the Command History, for more information.
–WALLPAPER **[*filename*]**	Displays a list showing which commands are bound to which key combinations. If you include *filename*, the list is written to the specified file. You can spool the file to get a hardcopy listing of keybindings.
–WARNING_BRIEF ***prompt***	Sets the warning prompt within ECL to *prompt*. The section, Customizing Your Prompts, includes an example of how to reset your prompts.
–NO_WILD_ABBREV **–WILD_ABBREV**	Controls whether ECL expands abbreviations when completing a pathname. With the default –NO_WILD_ABBREV, ECL does not expand abbreviations when you complete a pathname with the **expand_wild** command (Ctrl I). –WILD_ABBREV causes ECL to expand abbreviations and global variables when completing a pathname.
–WILD_DIRECTORY **–NO_WILD_DIRECTORY**	Controls whether ECL appends the > character to any entry that is a directory name when a pathname is expanded. With the default –WILD_DIRECTORY, > is appended whenever ECL expands an incomplete pathname into a directory name. –NO_WILD_DIRECTORY prevents ECL from adding the > character.
–WILD_MENU **–NO_WILD_MENU**	Controls numbering of the menu generated by the **expand_wild** command (Ctrl I). With the default –WILD_MENU, the selections are numbered for easy selection with the **expand_wild_menu** command (Esc *n* Esc Ctrl I). –NO_WILD_MENU prevents ECL from numbering the selections.

8

Customizing Your Environment

This chapter introduces a variety of techniques for customizing your working environment. It shows how you can

- Change the form of the prompts displayed at your terminal
- Define short abbreviations for PRIMOS command lines and arguments
- Define global variables to stand for strings that you can use both at command level and in programs
- Create a special login file of PRIMOS commands that are automatically carried out every time you log in
- Send messages and set your terminal's ability to receive messages

The chapter also briefly introduces commands to set maximum limits, or quotas, on the amount of storage space allotted to directories on a disk.

Changing the Prompt Message

You can vary the PRIMOS prompts in two ways using the RDY command:

- You can tell PRIMOS to supply a **long prompt** that includes the time and a variety of other information.
- You can supply your own prompt message text.

Long Prompts

The display format for long prompts is

$$\left\{\begin{array}{l}\texttt{OK}\\ \texttt{ER}\\ \textit{text}\end{array}\right\}\ \textit{time CPU-time I/O}\ [\textit{command level}]$$

The prompt displays OK (for ready), ER (for error), or a user-defined *text*, followed by the time (*hh:mm:ss*, 24-hour clock), and the amount of CPU and I/O time used (in seconds) since the last prompt. The following examples show long ready, and long error prompt displays:

```
OK 16:23:25  0.024  0.021
ER 10:07:31  0.100  0.609
```

The long prompt also displays your command level if it is greater than 1. Command levels are explained in Chapter 12. To have PRIMOS display long prompts, use the –LONG option of the RDY command, as shown in the section, The RDY Command.

The Message Text

The OK, and ER! prompt messages are default values for both the short prompt and the message part of the long prompt. You can use the RDY command to change the text of the prompt message to any string with a maximum of 80 characters. This is especially useful if you work on several systems, since you can set a different prompt for each system. You can change the text both for the brief and the long forms of the prompts.

The RDY Command

The format for the RDY command is

RDY [*options*]

The options are summarized in Table 8-1.

If given without options, the RDY command displays a single long-form prompt.

If you use one of the options that creates your own prompt message, the text can have a maximum length of 80 characters. If the text contains any special characters or imbedded blanks, you must enclose it within single quotation marks.

You can use more than one option with the same RDY command.

The following example illustrates the use of RDY options:

```
OK, RDY -LONG
13:11:41  0.827  1.739
RDY -OFF
RDY -ON
13:11:56  0.066  0.000
RDY -RL 'Absolutely right!'
Absolutely right! 13:12:01  0.054  0.000
RDY -BRIEF -RB 'GO!'
GO!
```

TABLE 8-1
Options for the RDY Command

Option	*Function*
–LONG	Sets the terminal to the long form of prompt.
–BRIEF	Sets the terminal to the brief form of prompt.
–OFF	Suppresses prompts entirely.
–ON	Reactivates most recently set prompts after they have been turned off with the –OFF option.
–READY_LONG ***text***	Changes the long ready message to *text*. `OK,` is the default message at login time. The time and usage data that appears in the long prompt is not affected by changes to the message text.
–READY_BRIEF ***text***	Changes the brief ready message to *text*. `OK,` is the default message at login time.
–ERROR_LONG ***text***	Changes the long error message to *text*. `ER!` is the default text at login time.
–ERROR_BRIEF ***text***	Changes the brief error message to *text*. `ER!` is the default message at login time.
–WARNING_LONG ***text***	Changes the long warning message to *text*. `OK,` is the default text at login time.
–WARNING_BRIEF ***text***	Changes the brief warning message to *text*. `OK,` is the default message at login time.

Warning Prompts

The default `OK,` prompt is actually two prompts in one. Normally, the `OK,` prompt indicates that PRIMOS has carried out a command successfully and is waiting for further input. This is called the **ready prompt**. When PRIMOS cannot carry out a command or a program does not run to completion, you see the **error prompt** (the default is `ER!`). But there are times when programs run to completion but issue warning messages, because they have encountered unexpected conditions. In these situations, PRIMOS issues a **warning prompt**. By default, PRIMOS does not distinguish between ready and warning messages; it uses the `OK,` prompt for both. You can use the RDY command to have PRIMOS distinguish between the two in one of three ways:

- Change your ready prompt (using the –READY_LONG and/or –READY_BRIEF options), but not your warning prompt; the ready prompt is changed to your new text, and the warning prompt is the default `OK,`.
- Change only the warning prompt (using the –WARNING_LONG and/or –WARNING_BRIEF options).
- Change both ready and warning prompts to different messages.

Notes

Any changes you make using the RDY command are in effect only during the current work session. If you log out and log in again, you see the default prompts. If you want the changes to be in effect everytime you log in, incorporate a RDY command in your login command file, discussed later in this chapter.

By default, ECL uses whatever prompts are in effect when you invoke ECL. If you set new prompts using the PRIMOS RDY command, ECL displays them. You can also set your own ECL prompts. See the section, Customizing Your ECL Prompts, in Chapter 7.

Creating and Using Abbreviations

You can use the ABBREV command to create abbreviations for frequently used command lines and arguments. For example, if you create the abbreviation NEWBOOK to stand for the pathname BOOKS>FICTION>MYSTERY>MANUSCRIPTS>EDITED>AUGUST, then you can attach to this directory with the short command

```
OK, ATTACH NEWBOOK
```

The ABBREV Command

You use the ABBREV command to create abbreviations and make them available to PRIMOS. There are three steps to this procedure, each of which is carried out with one of the ABBREV command options:

1. Create an empty abbreviation file.
2. Define abbreviations within the file.
3. Activate the file during any work session in which you want to use the abbreviations.

The general format of the ABBREV command is

$$\textbf{ABBREV}\begin{Bmatrix}[\textit{pathname}]\ [\textit{options}]\\ \textbf{options}\end{Bmatrix}$$

The ABBREV command takes several options. The ones you need to get started are discussed in this section. For a complete list of options and their uses, see the *PRIMOS Commands Reference Guide.*

Creating an Abbreviation File

To create and activate a new abbreviation file, use the ABBREV command with the –CREATE option:

ABBREV ***pathname*** **–CREATE**

where *pathname* names the file to be created. If the file is in the current directory, you can use a filename alone.

For example, assume that you are attached to a directory named MYDIR. The following command creates and activates an abbreviation file named MY.ABBREV in MYDIR:

```
OK, ABBREV MY.ABBREV -CREATE
```

Note

The ABBREV command with the -CREATE option activates the abbreviation file it creates. If another abbreviation file is currently active when you issue this command, it becomes inactive. While the new file is active, the only abbreviations available to you are the ones it contains. Reactivating an existing abbreviation file is discussed below.

Defining Abbreviations

You may define abbreviations and put them into an abbreviation file by using the -ADD_COMMAND, -ADD_ARGUMENT, and -ADD options of the ABBREV command. The format is

$$\textbf{ABBREV}\ [\textit{pathname}] \left\{ \begin{array}{l} \textbf{-ADD\_ARGUMENT} \\ \textbf{-ADD\_COMMAND}\ \textit{name value} \\ \textbf{-ADD} \end{array} \right\}$$

Argument	*Meaning*
pathname	The abbreviation file to which the abbreviation is added. This file also becomes the active abbreviation file. If *pathname* is omitted, the abbreviation is added to the abbreviation file currently active for your user ID.

Note

With all options of the ABBREV command, if you give a pathname or filename argument, the file specified is activated. If you give the ABBREV command without a pathname or filename, the currently active file is affected.

name	The abbreviation name. Abbreviation names can have as many as eight characters, but cannot include spaces, single quotation marks ('), commas, right angle-brackets (>), or vertical bars (|).
value	The string that the abbreviation name stands for. This can be any character string, including commands, arguments, and options, that can appear in a PRIMOS command line. The use of variables in abbreviations is discussed below.

Which option you use depends on how you want to use the abbreviation.

The -ADD_COMMAND Option: The -ADD_COMMAND option adds an abbreviation that is expanded to its full form only when it occurs in the command position of a command line. The first element of a command line after the prompt occupies the command position. For example,

```
OK, ABBREV -ADD_COMMAND TRM TERM -DISPLAY
```

adds the abbreviation TRM to the currently active abbreviation file, and defines it as standing for the command line

```
TERM -DISPLAY
```

Whenever this abbreviation file is active, you can get information on the current TERM settings by typing

```
OK, TRM
```

The –ADD_ARGUMENT Option: The –ADD_ARGUMENT option adds an abbreviation that is expanded to its full form only when it occurs in an argument position in a command line. An element is in an argument position when it occurs after the command position. For example,

```
OK, ABBREV -ADD_ARGUMENT MYBOOK  BOOKS>FICTION>MSS>MINE
```

adds the abbreviation MYBOOK to the currently active abbreviation file. If you use MYBOOK as a command line argument while the abbreviation file is active, it is expanded to BOOKS>FICTION>MSS>MINE. For example,

```
OK, DELETE MYBOOK
```

is equivalent to

```
OK, DELETE BOOKS>FICTION>MSS>MINE
```

The –ADD Option: The –ADD option adds an abbreviation that is expanded to its full form when it occurs anywhere on the command line.

Although the –ADD option creates an abbreviation that expands as either a command or as an argument, the more specific options –ADD_COMMAND and –ADD_ARGUMENT are usually preferred, because they avoid errors that can occur if, for example, you mistakenly place a command abbreviation in an argument position.

Abbreviations for Multiple Commands

You can define an abbreviation that stands for a series of commands by separating the commands with semicolons (;), just as you specify a series of commands in a PRIMOS command line interactively. For example,

```
OK, ABBREV -ADD_COMMAND AL ATTACH BOOKS>MSS; LD
```

defines an abbreviation AL that attaches you to the directory BOOKS>MSS and lists the directory's contents.

Note

Because the ABBREV command interprets a semicolon as the beginning of another command line to be included in the abbreviation, you cannot give two ABBREV commands on the same command line. That is, the command

```
OK, ABBREV -AC AL ATTACH MYDIR; LD; ABBREV -DELETE HOME
```

does not first create an abbreviation called AL and then delete one called HOME. Instead, it creates an abbreviation called AL that stands for the following command line:

```
ATTACH MYDIR; LD; ABBREV -DELETE HOME
```

Every time you execute this abbreviation, it attempts to delete an abbreviation called HOME from the active abbreviation file.

Using Variables in Abbreviations

You can include variables in your abbreviations. This often makes abbreviations more generally useful. For example, suppose you want to be able to attach to *any* directory and list its contents using a single command. You can do this by rewriting the AL abbreviation above using a variable in place of the pathname BOOKS>MSS. The following command accomplishes this:

```
OK, ABBREV -ADD_COMMAND AL ATTACH %1%; LD
```

The variable in this case is designated by `%1%`. Now, when you type a command line beginning with the abbreviation AL, whatever you type after the abbreviation is substituted for the variable %1% when the abbreviation is expanded. For example, when you type the command line

```
OK, AL ACCOUNTS>PAYABLE
```

the abbreviation processor substitutes the pathname ACCOUNTS>PAYABLE for the variable %1%. Your command line is, therefore, equivalent to

```
ATTACH ACCOUNTS>PAYABLE;LD
```

In effect you have created a new PRIMOS command with the format

AL ***pathname***

that attaches you to any directory you specify and lists its contents.

Multiple Variables: You can define a maximum of nine variables in a single abbreviation, using the format *%number%* for each variable. *number* indicates which string from the command line is to be substituted for each variable. The symbol %1% tells the abbreviation processor to substitute the first string typed after the abbreviation; %2% stands for the second string; and so on. For example, suppose you define an abbreviation with the command

```
OK, ABBREV -ADD_COMMAND CLD COPY %1% %2%; LD %2%
```

Now, if you give the command line

```
OK, CLD PAYMENTS ACCOUNTS>AUGUST
```

the abbreviation processor substitutes the string PAYMENTS for the variable %1% and ACCOUNTS>AUGUST for both occurrences of the variable %2%. Your command line is therefore equivalent to

```
COPY PAYMENTS ACCOUNTS>AUGUST; LD ACCOUNTS>AUGUST
```

When it processes your command, PRIMOS copies a file called PAYMENTS from the current directory to the directory ACCOUNTS, names the new file AUGUST, and then shows you the directory listing of the new file. In effect, you have created a command to copy a file and list the new copy. Your command has the format

CLD ***old_pathname new_pathname***

Activating an Abbreviation File

In order to use the abbreviations in an abbreviation file, the file must be active. When you create a new abbreviation file it automatically becomes the active file. If you add an abbreviation to an existing abbreviation file that is not currently active, the file also becomes active. If you want to activate an existing abbreviation file without adding any abbreviations, use the ABBREV command in one of the following formats:

ABBREV ***pathname*** [**–ON**]

or

ABBREV –ON

- You can always activate an abbreviation file using the first format. *pathname* specifies the abbreviation file to be activated. You can use a filename alone if the file is in the current directory. The –ON option may be omitted.
- You can use the second form if an abbreviation file has previously been activated during the current work session but no abbreviation file is currently active. This reactivates the most recently active abbreviation file.

Once your abbreviation file has been activated, it remains active until you either activate a different abbreviation file, log out, or deactivate it.

You can deactivate the currently active abbreviation file with the –OFF option:

ABBREV –OFF

Deleting Abbreviations

Use the –DELETE option of the ABBREV command to delete abbreviations. The format is

ABBREV [***pathname***] **–DELETE** ***name-1*** [**...*name-n***]

Argument	*Meaning*
pathname	Specifies the file from which the abbreviations are to be deleted. If the file is in the current directory, you can give the filename alone. If the file is not currently active, it is activated. If you don't specify a pathname, the abbreviations are deleted from the currently active file.
name-1 **[*...name-n*]**	Specify the names of the abbreviations you want to delete. You can use wildcards to select several similar abbreviations.

Listing Abbreviations

Use the –LIST option to display the contents of an abbreviation file. The format is

ABBREV [*pathname*] –LIST [***name-1*** [***...name-n***]]

Argument	*Meaning*
pathname	Specifies the abbreviation file to be listed. If the file specified is not currently active, it is activated. If *pathname* is omitted, the –LIST option lists the currently active abbreviation file.
[***name-1*** [***...name-n***]]	Optionally specify the names of abbreviations you want listed. You can use wildcards. If you omit the list of names, the entire file is listed.

In the listing, abbreviations added with –ADD_COMMAND are preceded in the listing by (C), those added with –ADD_ARGUMENT by (A).

WARNING

Do not try to list an abbreviation file with SLIST, or edit one with a text editor like ED or EMACS. Abbreviation files are not text files. Trying to list one with SLIST can produce unexpected results at your terminal. Editing an abbreviation file with a text editor can damage the file and make it unusable.

Using Abbreviations

You can use the abbreviations from an active abbreviation file in any command given to PRIMOS interactively. You just substitute the abbreviation name for the command line text it stands for. When an abbreviation file has been activated, the PRIMOS abbreviation processor scans each command line you type at the terminal, checking each word against the active abbreviation file. If it encounters any abbreviations, they are expanded to their full form before the command is passed to the command processor.

Note

You can see the expanded version of any abbreviation you include in a PRIMOS command line by giving the command

```
OK, ABBREV -VERIFY
```

This causes PRIMOS to redisplay all command lines, with any abbreviations expanded, before they are executed.

You can also use abbreviations in CPL programs (CPL is introduced in Chapter 15). You cannot use abbreviations in COMINPUT files. (COMINPUT files are introduced in Chapter 14.)

Note

If you include **global variables** or **functions** in an abbreviation definition, you may need to use the PRIMOS syntax supression character, the tilde (~). This keeps PRIMOS from evaluating the variables or functions when the abbreviation is defined. The next section explains global variables. It includes an example of an abbreviation that uses a global variable, and it shows you how to use syntax suppression. Functions are discussed in the *PRIMOS Commands Reference Guide*.

Global Variables

PRIMOS provides several ways to use variables that stand for strings, such as pathnames and command arguments. The variables that are introduced with the ABBREV command take their values directly from a command line that you type at the terminal. You can also define variables that take their values from a file. Such variables are called **global variables**, and the file in which their values are defined is called a **global variable file**. They are said to be *global* because, once they have been defined, they are widely accessible. Global variables are available

- Interactively, at PRIMOS command level
- To abbreviations
- To CPL programs (see Chapter 15)
- To COMINPUT files (see Chapter 14)
- To programs written in some high-level languages

The process of defining and using global variables is similar to the process of defining and using abbreviations.

1. You create and activate a global variable file.
2. You define variables in the active file.
3. You use variables defined in the active global variable file to stand for argument strings in PRIMOS commands and programs.

Once a global variable has been defined, it is available whenever the file that defines it is active.

Global Variable Related Commands

The PRIMOS commands to create and activate global variable files and define global variables are described briefly below. For complete details, see the *CPL User's Guide* or the *PRIMOS Commands Reference Guide*.

Command	*Function*
DEFINE_GVAR ***pathname*** [**-CR**EATE]	Activates the global variable file *pathname*. With the -CREATE option, creates an empty global variable file and activates it.
SET_VAR ***name*** [:=] ***value***	Defines a new variable and places it in the active global variable file, or changes the value of an existing variable. *name* may contain a maximum of 32 characters. Names of global variables must begin with a dot (.), as in .ALPHA. *value* is an alphanumeric character string. The assignment symbol (:=) is optional.
LIST_VAR [*var_name...*]	Lists the variables and values contained in an active global variable file. You can specify *var_name...* to select specific variables for listing. You can use wildcards to specify several similar variables.
DELETE_VAR ***var_name...***	Deletes variable(s) from an active global variable file. You can use wildcards to select several similar variables.

Caution

Do not use a dollar sign ($) as the last character in global variable names that you define; this character is reserved for global variables defined by Prime.

The following example illustrates the use of these commands. Suppose that you want to create and activate a global variable file called VARFILE.GVAR. Use the DEFINE_GVAR command:

```
OK, DEFINE_GVAR VARFILE.GVAR -CREATE
```

Now suppose that you want to define variables to stand for the pathnames of two directories that you often use. Do this with the SET_VAR command:

```
OK, SET_VAR .REPORT ACCTS>PAYMENTS>FY87
OK, SET_VAR .MEMO MYDIR>PAPERWORK>MEMOS
```

You can check that the variables have been properly defined with the LIST_VAR command:

```
OK, LIST_VAR
.REPORT                     ACCTS>PAYMENTS>FY87
.MEMO                       MYDIR>PAPERWORK>MEMOS
```

Using Global Variables

Global Variables in Interactive Commands: You can use global variables interactively by including them in a command line. You specify a global variable in a command line as *%name%*, where *name* is a variable defined in the currently active global variable file. You normally use variables to stand for arguments such as long pathnames, although a global variable can stand for a whole command line, much like an abbreviation.

For example, if VARFILE.GVAR is currently activated, you can attach to the directory ACCTS>PAYMENTS>FY87 by giving the following command at the terminal:

```
OK, ATTACH %.REPORT%
```

Global Variables in Abbreviations: You can include global variables in an abbreviation definition in much the same way that you include variables that take their values from the command line. You use the same format as you do when specifying a global variable interactively: *%name%*.

If you use a global variable in an abbreviation, you must be sure that PRIMOS interprets it as you intend. Suppose you have defined the global variable .BOOK to stand for the pathname MSS>EDITING>NEW in the currently active global variable file. If you now define an abbreviation with the following command,

```
OK, ABBREV ADD_COMMAND BOOK ATTACH %.BOOK% ;LD
```

and then check your abbreviation file with ABBREV –LIST you find the following definition:

```
BOOK   ATTACH MSS>EDITING>NEW; LD
```

This is probably not what you intended. PRIMOS evaluated the global variable .BOOK when it executed the ABBREV command, replacing it with its current value (MSS>EDITING>NEW) in the abbreviation definition. This means that if you later change the value of .BOOK in your global variable file, the BOOK abbreviation still attaches you to the old directory.

Usually, what you want is to include the variable itself in the abbreviation definition. You do this with the **tilde** (~), the PRIMOS **syntax suppression** character introduced in Chapter 6. When you put a tilde at the beginning of the ABBREV command line, PRIMOS does not evaluate global variables when it defines the abbreviation. For example, if you give the command line

```
OK, ~ABBREV ADD_COMMAND BOOK ATTACH %.BOOK%;LD
```

then the book abbreviation is listed as

```
BOOK   ATTACH %.BOOK%; LD
```

Now the variable .BOOK is evaluated each time the BOOK abbreviation is expanded. In other words, when you type BOOK in a command line, PRIMOS attaches you to whatever directory is currently defined as .BOOK in your global variable file. This means that when you change the value of .BOOK in the currently active global variable file, the BOOK abbreviation attaches you to the new directory.

Global Variables in Programs: CPL programs define global variables with the &SET_VAR directive or with the SET_VAR command (but *not* with the &ARGS directive). Variables are referred to in the same form as above: *%name%*. Programs written in high-level languages define global variables through the GV$SET routine, and refer to them through the GV$GET routine. The *CPL User's Guide* and the Prime language reference guides contain details.

Global Variables in the Batch and Phantom Environments: If you use global variables in a command file or program that runs as a Batch job or phantom, you must activate the appropriate global variable files *within* the command file or program. The Batch and phantom environments are described in Chapter 16.

Creating Login Files

PRIMOS can do much of the work of customizing your environment automatically, every time you log in. For example, you may want to establish certain terminal characteristics (using TERM), set prompts (with RDY), and activate abbreviation and global variable files (using ABBREV and DEFINE_GVAR) each time you log in. If your system has electronic mail or a bulletin board system, you might also want to give the commands to check your mail and read any messages. You can do all of these things using a login file.

A **login file** is a program or command file that PRIMOS automatically executes each time you log in. The file is usually written in PRIMOS Command Procedure Language (CPL). CPL is introduced in Chapter 15, but a simple CPL program can be just a series of PRIMOS commands stored in a file with the filename suffix .CPL. You can write a simple login CPL file in two steps:

1. Use ED or EMACS to create a file containing the PRIMOS commands you want executed each time you log in. Enter one command per line in your file.
2. Save this file in your origin directory with the filename LOGIN.CPL.

You can also write a login file in any programming language supported by PRIMOS or you can use a command input file. These options are described briefly below. Every time you log in, PRIMOS looks for a login file in your origin directory. If such a file is found, PRIMOS executes it automatically before beginning its interactive dialog with you.

A Sample LOGIN.CPL

The following example shows a LOGIN.CPL file created with ED:

```
TERM -ERASE ^210 -KILL ^377 -XOFF
ABBREV MY.ABBREV -ON
RDY -READY_BRIEF Hello
DEFINE_GVAR MY.GVAR
SLIST NOTES
```

The command lines have the following meanings:

TERM –ERASE ^210 –KILL ^377 –XOFF: This entry uses the TERM command with three options to set new erase and kill characters and enable the use of Ctrl S and Ctrl Q to stop and start screen output. The options are

Option	*Meaning*
–ERASE ^210	Sets the erase character to Backspace on a Prime terminal. ^210 is a special notation for entering Backspace in a file you create with ED. You enter a nonprinting character with ED by giving the character's octal ASCII code in the following format: ^*octal number*. You can find the octal number for any character in Appendix C, The Prime Extended Character Set. In this case, 210 is the octal code for Backspace. **Note** You use the ^*octal number* format to enter a nonprinting character in a file using ED. If you list the file using SLIST, ^*octal number* does not appear in the listing. Instead, the nonprinting character appears as a blank or as some other symbol, depending on the terminal. If you are creating a CPL file with EMACS, consult the EMACS documentation to find out how to enter nonprinting characters. For more information on the representation of nonprinting characters in PRIMOS, see Appendix C.
KILL ^377	Sets the kill character to Delete on a Prime terminal.
–XOFF	Allows you to use Ctrl S and Ctrl Q to switch terminal output on and off. See Chapter 1 for further information on this and other options of the TERM command.

ABBREV MY.ABBREV –ON: This entry activates an abbreviation file in the origin directory called MY.ABBREV. By activiating an abbreviation file with the LOGIN.CPL, you can make a set of abbreviations automatically available for the subsequent work session.

Note

While your LOGIN.CPL is executing, you are attached to your origin directory. Therefore, a filename like MY.ABBREV refers to a file in the origin directory. Similarly, if you give a relative pathname as an argument to any of the commands in your LOGIN.CPL, it refers to a path relative to your origin directory.

RDY –READY_BRIEF Hello: This entry changes the short version of the ready prompt from `OK,` to `Hello`.

DEFINE_GVAR MY.GVAR: This entry activates a global variable file in your origin directory called MY.GVAR.

SLIST NOTES: This entry lists a file called NOTES in your origin directory. If you maintain a file of reminders and messages, using SLIST in your login file can be a convenient way to have them appear on the screen automatically when you log in.

If you store this LOGIN.CPL in your origin directory, PRIMOS executes the commands it contains every time you log in. When you log in, you see any output from the commands in the LOGIN.CPL file before PRIMOS puts the ready prompt on the screen for the first time. In the example, the first commands produce no screen output. After they execute, you see a listing of the contents of NOTES. Finally, PRIMOS displays the new ready prompt

```
Hello
```

indicating that you are at command level, and that PRIMOS is ready to accept commands.

Alternate Forms for the Login File

Instead of creating a LOGIN.CPL file, you can create a login file that is a command input file or a program in one of the languages supported by PRIMOS. The file must be called one of the following:

LOGIN.COMI
LOGIN.RUN
LOGIN.SAVE

LOGIN.COMI: You can create a **command input file** called LOGIN.COMI as a login file. A command input file is a list of PRIMOS command lines, stored in a file with the filename suffix .COMI. Command input files are explained in detail in Chapter 14.

For simple login files like the one given in the previous section, there is not much difference between a LOGIN.COMI and a LOGIN.CPL. Command input files are more limited than CPL files, however. For example, a LOGIN.COMI cannot contain any commands that require user input. LOGIN.CPL is usually preferable because the CPL programming language gives you more control over the execution of PRIMOS commands, allows user input, allows you to use abbreviations, and offers many other features not available in command input files.

LOGIN.RUN and LOGIN.SAVE: You can create a login file called either LOGIN.RUN or LOGIN.SAVE using one of the programming languages supported by PRIMOS. If you link your program with BIND, then your login file is an EPF called LOGIN.RUN. If you use LOAD, your login file is a static-mode executable file called LOGIN.SAVE. For more information on compiling and linking programs see Chapters 10 and 11.

Sending Messages

Use the MESSAGE command to send or receive one-line messages. Messages may be sent from

- Any user terminal to any other user terminal
- Any user terminal to the supervisor terminal (for messages to the operator)
- The supervisor terminal to all users
- The supervisor terminal to a specified user

User Messages

The format of a user-to-user or user-to-operator message is

$$\text{MESSAGE} \begin{Bmatrix} \textit{user-ID} \\ \textit{-usernumber} \end{Bmatrix} [\text{-NOW}]\ [\text{-ON } \textit{systemname}]$$

one-line text of message

Argument	*Meaning*
user-ID ***–usernumber***	Identify the recipient of the message. You must include the hyphen before *–usernumber*. To get a list of user IDs and user numbers for logged in users, give the STATUS USERS command (explained in Appendix G) or the MESSAGE –STATUS command (explained in the next section, Querying Receive States). To send a message to the supervisor terminal, omit *user-id* or *–usernumber*. If you send a message to a *user-id*, all users logged in as that name receive the message. If you send a message to a *-usernumber*, only the single user with that number receives the message.
systemname	To send a message to a user logged in to another system in your network, specify *user-id* or *–usernumber* together with the –ON *systemname* option. *systemname* is the nodename of the other user's system. Use the STATUS NETWORK command to get a list of nodenames for your network.
message text	A single-line message with a maximum of 80 characters. Only printing characters and Ctrl G (bell) are sent. If you include erase of kill characters, the line is edited before it is sent.

Sending a message produces two lines of information on the receiver's terminal. The top line contains information about the sender; the second contains the text of the message. The format is

```
*** sendername (user number [on systemname]) at hh:mm
one-line text of message
```

sendername is the sender's user ID. *number* is the number of the sender's terminal. *systemname* is the name of the system the sender is using. *systemname* appears only if you are working on a networked system. *hh:mm* is the time of day in hours and minutes that the message was sent (24-hour clock). For example, if user BEECH on system SY3 gives the command

```
OK, MESSAGE MAPLE
Hello to Maple.
```

user MAPLE on SY3 sees the following message display:

```
*** BEECH (user 66 on SY3) at 15:16
Hello to Maple.
```

If you specify the –NOW option when you send a message, the message is displayed immediately on the receiver's terminal.

If you don't specify the –NOW option, the message is displayed when the receiver returns to PRIMOS command level.

Setting Receive States

To control the flow of messages, you can set your terminal's **receive state** with the MESSAGE command. Use the MESSAGE command with one of the following options.

Option	*Function*
–ACCEPT	Enables reception of all messages. This is the default state.
–DEFER	Inhibits immediate messages. Messages sent to you with the –NOW option are rejected. You receive messages sent without the –NOW option when you return to command level.
–REJECT	Inhibits all messages.

Deferring or rejecting messages is useful when you do not want messages to interrupt a terminal session. Otherwise, messages can disrupt the screen display while you are doing other work.

You cannot send a message while you are in MESSAGE –REJECT mode, and you cannot send an immediate message (using the –NOW option) while you are in MESSAGE –DEFER mode. These restrictions exist because the receiver cannot respond to your message.

Querying Receive States

You can find out the receive state of a user's terminal with the –STATUS option of the MESSAGE command.

The format is

$$\textbf{MESSAGE -STATUS} \left\{ \begin{array}{l} \textit{\textbf{user-id}} \\ \textit{\textbf{usernumber}} \\ \textbf{ME} \end{array} \right\} [\textbf{-ON systemname}]$$

The different formats list the following information:

Command	*Function*
MESSAGE –STATUS	Lists the receive state of all users on your system.
MESSAGE –STATUS ***user-id***	Lists the receive state of all users with the name *user-id* on your system.
MESSAGE –STATUS ***usernumber***	Lists the receive state of the user with the number *usernumber* on your system.
MESSAGE –STATUS ME	Lists the receive state of your own terminal.

To find the receive state of a user or users logged in to another system in your network, specify the –ON *systemname* option with any of the first three command lines listed above.

Disk Quotas

To ensure equitable sharing of disk storage, System Administrators and users can set limits (called **quotas**) on the amount of storage space that directories can occupy on a disk. You can set a quota on a directory if you have the appropriate access rights. To set a directory quota you need Protect (P) access rights (for systems that use ACLs) or Owner (O) rights (for systems that use directory passwords) to the next higher directory. Normally, this means that only the System Administrator can set or change quotas on top-level directories. Individual users can set or change quotas on their own subdirectories if they want to provide personal checks on their own storage use. The commands for using quotas allow users to

- Set a maximum storage quota on a subdirectory (SET_QUOTA)
- Change an existing quota (SET_QUOTA)
- Examine existing quotas and current storage use (LIST_QUOTA, LD, SIZE)

For more information on these commands, see the *PRIMOS Commands Reference Guide.*

Part II: Programming

9

Introduction to PRIMOS Programming

The chapters in Part II introduce you to the programming tools available in PRIMOS. This chapter provides brief overviews for both experienced and new programmers.

For Experienced Programmers

The PRIMOS programming tools provide a complete development environment for programming in a wide variety of languages. You can

- Use ED or EMACS to write and edit source code. If you are using EMACS, be sure to learn about language modes. These enable you to compile and debug programs without leaving the editor and help you produce correctly formatted source code. (See Chapter 4 and the *EMACS Reference Guide.*)
- Compile your source code with one of the compilers supported by PRIMOS. (See Chapter 10.)
- Use the PRIMOS linking utilities BIND, SEG, or LOAD to produce executable runfiles. (See Chapter 11.)
- Execute runfiles interactively using the RESUME command. (See Chapter 12.)
- Debug you programs with DBG, the Prime Source Level Debugger. (See Chapter 13.)
- Examine and modify your command environment using a variety of PRIMOS commands. (See Chapter 12.)

For more detailed information, consult the the Prime documentation for the language you are using.

Introductory Overview

If you are learning to program on a Prime system, the following sections provide a brief overview of the programming process, which is discussed in detail in Chapters 10-13.

The Programming Process

The four basic steps to writing and running a program on a Prime computer are

1. Writing the program **source code** using a text editor such as ED or EMACS
2. Compiling the program using one of the PRIMOS supported **compilers**
3. Linking the program using one of the PRIMOS **linking utilities**
4. Running the program

Since programs seldom run as expected the first time, you often need to debug them as well. Prime provides the DBG utility to aid you in debugging programs.

Creating the Source Code

Consult the Prime documentation for the language you are using to find out where the Prime version differs from standards or other versions you may be familiar with. Although most languages are standardized, every implementation differs slightly. This is especially true in the case of input, output, and file operations, which tend to be operating system dependent.

Use a text editor to write your program. If you use EMACS, learn about the EMACS language modes. These allow you to test, compile, and modify programs in some languages without leaving the editor. They also help you to produce correctly formatted source code.

The file that you create with the text editor is called the **source code**. It is a text file containing program statements; it cannot be run by the computer in this form. First you must transform it into an executable file by carrying out two steps: compiling and linking.

Compiling Your Program

Compiling takes a source code text file in any of the Prime supported languages and creates a new file called an object file. The **object file** contains your program in a generalized form that can later be linked to create an executable program. You compile your program using a utility called a **compiler**.

Each Prime supported language has its own compiler. General information about the compilers is given in Chapter 10. For specific information about the compiler for the language you are using, consult the Prime documentation for that language.

Linking Your Program

Use one of the PRIMOS linking utilities to turn your object file into an program the computer can execute, called an **executable file** or **runfile. Linking** ties your object file to libraries and other object files that it needs to execute.

In general, your object file needs to be linked to one or more libraries before it can run. A **library** is a standard subprogram that carries out certain functions, such as input and output operations, for your program. Consult the documentation for the compiler you are using to find out which libraries you need to link to your program.

Besides linking libraries to your object file, you can also link together several of your own object files to create a single executable program. This makes it much simpler to develop and modify large programs, because you can write, compile, and test the program a section at a time.

You can write different parts of a program in different languages. The object files created by different language compilers are compatible. As a result, you can write different parts of a program in different languages, compile them with the appropriate compilers, and then link the object files together into a single program.

Linking Utilities: PRIMOS provides three linking utilities: BIND, LOAD, and SEG. Each utility creates a different kind of executable file.

The prefered linking utility is BIND. This creates an executable file called an **Executable Program Format** (**EPF**). EPFs are more flexible than the executable files created by the other linking utilities, because they are **dynamic**; the computer can execute them in any available memory space. Because the computer can locate EPFs wherever memory is available, it can simultaneously keep several EPFs in memory without conflict. This means, for example, that you can begin to execute one EPF, halt its operation, execute a second EPF or PRIMOS commands, and then resume operation of the first EPF wherever it was halted.

The executable files produced by LOAD are **static**. PRIMOS can execute static-mode programs in a single area of memory only, and can keep only one static-mode program in memory at a time. As a result, static-mode programs cannot be stopped and started as freely as EPFs.

(The SEG linker is not covered in this book. See the *SEG and LOAD Reference Guide* for more information.)

Note

The choice of linking programs can introduce one complication. Your object file must be compiled in an addressing mode that is compatible with the linker you use. The **addressing mode** determines how the compiler calculates memory addresses in your program. If you use BIND, this is not a problem, because all the compilers produce a compatible addressing mode by default. If you aren't using BIND, you may need to specify another addressing mode. Chapter 10 explains how.

Running Your Program

Once you have linked your program to create an executable file, it is ready to run. You run programs interactively by giving the PRIMOS **RESUME** command. The details of this process are given in Chapter 12.

Debugging Your Program

Since most programs don't run as expected the first time, you usually have to modify, recompile, link, and attempt to run your program several times. To help you with this process, PRIMOS provides a **debugging utility** called **DBG**. DBG allows you to keep track of your program's operation while you are testing it. You can, for example, halt the program at predetermined points and examine the values of variables to be sure they are behaving as expected. DBG's features are summarized in Chapter 13.

More Information

You can do most simple programming on a Prime computer with two sources of information:

- The Prime documentation for the language compiler that you are using
- Chapters 10-13 of this book

Remember, however, that the Prime language documentation describes the Prime implementation of each language. Prime language documentation generally assumes that you already have some familiarity with programming in the language documented. If you are beginning to use a language for the first time, a standard text for that language is also helpful. You can then use the Prime documentation to learn where the Prime implementation differs from the standard. The Prime documentation also gives you language-specific information on the compiling and linking processes.

10 Compiling Programs

Compiling is the first step in a two part process that transforms source files written in high-level programming languages into programs that can be run by the computer:

1. You compile your source code using one of the PRIMOS compilers. The compiler creates a file containing your program in a generalized form called an **object file** (sometimes also called a **binary file**).
2. You then link the object file using one of the PRIMOS **linking utilities** to create an **executable file** that can actually be run by the computer.

This chapter shows you how to compile programs. The linking step is discussed in Chapter 11.

Each of the programming languages supported by PRIMOS has its own compiler. The compilers are listed in Table 10-1.

This chapter discusses topics common to all compilers:

- Source files
- Compilers
- Object files
- Addressing modes
- Listing files and cross-reference files

The individual language guides cover details of specific Prime supported languages and their compilers.

The Source File

Use either the ED or EMACS text editor to create source files. If you use EMACS, learn about the EMACS language modes. For many of the Prime supported languages, the language modes allow you to edit, compile, and debug your program without leaving the editor. Compiler error messages appear in one window of your screen while the offending

TABLE 10-1
PRIMOS Language Compilers

Compiler	*Language*
CBL	COBOL 74
CC	C
F77	FORTRAN 77
FTN	FORTRAN IV
PASCAL	Pascal
PL1	PL/I
VRPG	RPG II (V-mode compiler)
PMA	Assembly

source file text appears in the other. EMACS language modes also provide several tools to help you produce correctly formatted source code. See the *EMACS Reference Guide* for information on the EMACS language modes.

Refer to Prime documentation for the language you are using to find out where the Prime implementation differs from standard versions. This is especially important in the case of input, output, and file operations, which are often operating system dependent. If you want your programs to call PRIMOS subroutines directly, consult the *Subroutines Reference* series for detailed information.

Filename Conventions

The name you choose for your source file should use one of the standard **compiler name suffixes** listed in Table 10-2. For example, a FORTRAN IV program with the basename DRAGON should be named DRAGON.FTN, while a FORTRAN 77 program with a basename WYVERN should be named WYVERN.F77. These suffixes are the defaults expected by the compilers when they are invoked. Using these suffixes also helps you keep track of your program files and maintains consistency with other filenames on the system.

Invoking Compilers

Invoke a compiler from PRIMOS command level with the following command:

compiler pathname [*options*]

Argument/Option	*Meaning*
compiler	Specifies the name of the compiler for the language in which your source program is written. Names for currently available compilers are listed in Table 10-1.
pathname	Specifies the pathname of your source program. If the source program is in your current directory, you can use an objectname alone.
options	Let you control • The creation of output files • Addressing modes • The debugger interface

TABLE 10-2
Recognized Language and Compiler/Linker Filename Suffixes

Suffix	*Meaning*	*Recognized by*	*Supplied by*
.BASIC	BASIC/VM source file	BASIC/VM compiler	User
.CBL	COBOL 74 source file	CBL compiler	User
.CC	C source file	CC compiler	User
.F77	FORTRAN 77 source file	F77 compiler	User
.FTN	FORTRAN IV source file	FTN compiler	User
.PASCAL	Pascal source file	PASCAL compiler	User
.PL1	PL/I source file	PL1 compiler	User
.PMA	PMA source file	PMA assembler	User
.RPG	RPG II source file	VRPG compiler	User
.VRPG	RPG II source file	VRPG compiler	User
.BIN	Object file (created by compiler)	BIND, SEG, LOAD	Compilers or user
.LIST	Listing file (created by compiler)		Compilers or user
.RUN	Runfile created by V-mode linker, BIND	BIND, RESUME	BIND or user
.SAVE	Runfile created by R-mode linker, LOAD	LOAD, RESUME	LOAD or user
.SEG	Segment directory created by SEG	SEG	SEG or user

Specifying the Source Filename

Each compiler recognizes one of the standard filename suffixes by default. If you have used the standard suffix for your source filename, you need not specify the suffix when invoking the compiler. When you invoke a compiler without giving a suffix to the pathname, the compiler looks first for a file called *pathname.suffix*, where *suffix* is the standard sourcename suffix for the compiler. If *pathname* with a standard suffix is not found, the compiler then looks for *pathname* without the identifying suffix. For example, typing FTN DRAGON causes the FTN compiler to look first for DRAGON.FTN. If it doesn't find DRAGON.FTN, it looks for a file called simply DRAGON.

Compiler Defaults

By default, all compilers

- Produce object files
- Do not produce listing or cross-reference files
- Generate 64V-mode code

Options to change these defaults and use the debugger interface are described in the following sections.

Object Files

By default, all compilers create **object** (or **binary**) files.

Object Filenames

The compilers supply default object filenames automatically. The default name of an object file depends on the filename of your source program. If you use a source filename with a standard compiler name suffix in the form *basename.suffix*, the default object filename is *basename*.BIN. For example, when you compile the source file PAYROLL.CBL, the CBL compiler produces an object file called PAYROLL.BIN. If the name contains no suffix, the default object filename is B_*filename*. For example, if you use the PASCAL compiler to compile the source file SORT, the object file is called B_SORT.

It is strongly recommended that the filenames of your source programs use standard compiler-identifying suffixes. All your object filenames then automatically have the .BIN suffix. All three linkers (BIND, SEG, and LOAD) expect object files with the default suffix .BIN, so compiling and linking are easier if you stick with the standard suffixes from the beginning.

The –BINARY Option

Use the –BINARY option either to create a nondefault object filename or to suppress creation of an object file altogether. This option also allows you to add the .BIN suffix to the filename of your object file, even if the filename of your source program does not contain a standard compiler-identifying suffix.

With the CC and FTN compilers, the following arguments are available for the –BINARY option:

Option/Argument	*Meaning*
–BINARY YES	Creates a object file with the default name
–BINARY NO	Does not create a object file
–BINARY *pathname*	Creates an object file called *pathname*

Use the following arguments and options with the CBL, F77, PASCAL, PLI and VRPG compilers:

Option/Argument	*Meaning*
–BINARY	Creates an object file with the default name
–NO_BINARY	Does not create an object file (Use **–NOBINARY** with the CBL compiler)
–BINARY *pathname*	Creates an object file called *pathname*

Addressing Modes

The **addressing mode** determines how the compiler calculates addresses in your program. Prime supported compilers are capable of generating code using a variety of addressing modes. Table 10-3, at the end of this section, shows the addressing modes available with each compiler. Choose a mode that is compatible with the linking utility you are planning to use. (Linking utilities are discussed in the next chapter.) It is recommended that you use the BIND linking utility, which requires code compiled in either of the I or V modes. In most cases, use 64V mode, which is the default for all the compilers.

V and I Modes

Use V and I addressing modes for single or multisegmented programs, which can be as large as 32MB. Programs in these modes can take full advantage of the virtual address space. You can link V-mode and I-mode code with the recommended BIND utility to produce EPFs.

Mode *Description*

64V The most generally useful mode. It takes full advantage of the virtual address space and allows you to use BIND to create EPFs.

32I Handles double precision floating-point arithmetic more rapidly than the other modes do. Therefore, it is the mode of choice for many mathematical calculations. To generate 32I-mode code, use the –32I option. The following example compiles the FORTRAN77 program CHEERS in 32I mode:

```
OK, F77 CHEERS -32I
```

32IX The F77 and CC compilers can generate 32IX-mode code, an enhanced 32I-mode code that gives improved performance when accessing large data objects, such as segment-spanning arrays or common blocks. Programs compiled with the –32IX option can be run on most Prime computers. Earlier machines may need to be upgraded. Check with you System Administrator or operator to see if your computer is able to use code compiled in 32IX mode.

Note

References in this book to I and 32I mode apply to both 32I and 32IX modes.

R Mode

R mode is an addressing mode used only for single segment programs. A single segment program cannot exceed 128K bytes. You must link R-mode code with the LOAD utility to produce a static mode executable file, which is not as flexible as an EPF.

TABLE 10-3
Compiler Addressing Modes

	Addressing Mode				
Compiler	*32I*	*32IX*	*64V*	*64R*	*32R*
CBL		X	X		
CC		X	X		
F77	X	X	X		
FTN			X	X	X
PASCAL	X		X		
PLI	X		X		
VRPG			X		

X indicates that a mode is available with a compiler.

Listing Files

Each compiler can create a file that lists the source program. Use the –LISTING option to create a listing file. Compiler specific options are available to expand these listings and add more information. When a listing file is created, it is named by the compiler in the same way as the object file, using the .LIST suffix instead of .BIN suffix. If your source file has a standard compiler name suffix, the listing file is called *basename*.LIST. For example, the listing file for DRAGON.FTN is DRAGON.LIST. If the source filename does not have a standard compiler name suffix, then the listing file is named L_*basename*.

For the CC and FTN compilers, the following arguments are available for the –LISTING option:

Option/Argument	*Meaning*
–LISTING [YES]	Creates a listing file with the default name
–LISTING NO	Does not create a listing file
–LISTING *pathname*	Creates a listing file called *pathname*
–LISTING TTY	Displays the listing file at your terminal
–LISTING SPOOL	Prints the listing file on a line printer (not available with the CC compiler)

For the CBL, F77, PASCAL, PLIG, and VRPG compilers, use the following options and arguments:

Option/Argument	*Meaning*
–LISTING	Creates a listing file with the default name
–LISTING *pathname*	Creates a listing file called *pathname*
–LISTING TTY	Displays the listing file at your terminal
–LISTING SPOOL	Prints the listing file on a line printer
–NO_LISTING	Does not create a listing file

Cross-references

The compilers can also create cross-reference listings that show where variables appear in the program and provide other useful information. Listings are different for each language; specific details are given in each compiler guide. The –XREF option generates a cross-reference listing and appends it to the program listing. (If you have not specified the –LISTING option, the –XREF option creates a program listing for you.)

Compiler Messages

If an interactive compilation completes successfully, a message to that effect is displayed at your terminal. (If the compilation is not interactive, the message may be written to a COMO file. See Chapter 14 for information on COMO files.) If compilation is not successful, error and/or warning messages indicate the line where the error occurred and the type of error. Some severe errors halt the compilation as soon as they are discovered. Others allow the compilation to proceed. Each compiler has its own error messages.

Error messages produced by the CBL, CC, F77, PASCAL, and PL1 compilers include explanatory comments. Error messages produced by the FTN compiler are explained in the *FORTRAN Reference Guide.*

Combining Languages in a Program

Because all high-level languages are compatible at the object code level and use the same calling conventions, programs compiled by the CBL, CC, F77, FTN, PASCAL, and PL1 compilers can call subroutines compiled by any of the other compilers as long as they have been compiled using compatible addressing modes. For example, a program written in COBOL 74 can call a subroutine written in FORTRAN 77, which can, in turn, use a utility subroutine written in PL/I. Procedures compiled by the high-level language compilers may also call, or be called by, procedures written in the Prime assembler language, PMA.

When combining programs written in different languages, observe the following precautions:

- Write all input and output routines in a single language.
- Be sure that there is no conflict in data types for variables being passed as arguments. For example, an INTEGER*2 in FORTRAN must be declared as FIXED BIN(15) in PL/I.
- Compiled code must use compatible addressing modes. In general, code compiled in one of the I or V modes can call code compiled in any of the I or V modes. R-mode code can only call and be called by R-mode code.
- You must observe some special restrictions when FORTRAN IV and FORTRAN 77 routines are linked. These are discussed in the *FORTRAN 77 Reference Guide.*

A complete discussion of passing data types between languages appears in *Subroutines Reference I: Using Subroutines.*

11 Linking Programs

Once you have compiled a program using one of the PRIMOS compilers, you must link it with one of the linking utilities provided by PRIMOS. These utilities process your object file to create a program, called an **executable file** or **runfile**, that can actually be run by the computer.

Prime offers three linking utilities: BIND, SEG, and LOAD. BIND and LOAD are discussed in this chapter. Topics covered are

- The advantages of the BIND linker
- The linking process
- Using BIND
- Using LOAD

The discussion in this chapter covers features common to linking programs written in all languages. For language specific information, consult the Prime language documentation for the language you are using. For more information on BIND and EPFs, consult the *Programmer's Guide to BIND and EPFs*. For information on SEG, consult the *PRIMOS Commands Reference Guide* and the *SEG and LOAD Reference Guide.*

Advantages of the BIND Linker

Always use BIND, unless you have to link an R-mode program. BIND is a friendlier linker than LOAD. You can run BIND either as an interactive subsystem, or from the command line. In interactive mode, BIND accepts subcommands one at a time. Alternatively, you can give all the BIND subcommands together on the command line when you invoke BIND.

BIND is especially recommended, because it creates dynamic runfiles, called **Executable Program Formats (EPFs)**. EPFs offer many advantages to the user.

The Advantages of EPFs

An EPF is not bound to any specific area of memory. When you invoke an EPF, PRIMOS can load it into any available area of memory.

- Several programs, even suspended programs, can exist in memory at one time without interfering with each other.
- A suspended EPF can often be restarted using the START command even if other programs have been invoked since it was suspended.
- EPFs can freely call and be called by other EPFs without concern that one program may overwrite another.
- You never have to share programs explicitly. PRIMOS automatically shares EPFs among users.
- You can create personal EPF libraries that can be dynamically linked to your programs.

In contrast, LOAD creates **static runfiles.** These runfiles are static because they must always be executed in the same area of memory. Whenever a new static-mode program is invoked, it overwrites any other static-mode program already in memory. Therefore, only one static-mode program can exist in memory at a time.

Linkers and Addressing Modes

You must compile your object file using the correct addressing mode for the linker you plan to use:

- BIND requires V-mode code or I-mode code
- LOAD requires R-mode code

Converting Static Programs to EPFs

You can usually convert old static-mode programs to EPFs by relinking or recompiling and relinking. You can convert most static-mode programs that were compiled in I mode or V mode and linked with SEG simply by relinking with BIND. Static-mode programs that were compiled in R mode and linked with LOAD, must be recompiled and relinked. Depending on the language, such programs may require some modification of the source code. For possible restrictions on conversion to EPFs, see the *Programmer's Guide to BIND and EPFs.*

The Linking Process

BIND and LOAD work quite differently, but they share some basic features. The basic task of a linker is to resolve the external references in your program. **External references** are calls to subroutines not included in your program code itself. These can include

- Separately compiled user-supplied subroutines
- Libraries of various types
- PRIMOS resources

User-supplied Routines

User-supplied subroutines are program modules that you have developed and compiled separately from the main program. The linker incorporates these routines in the executable file.

Libraries

Libraries are sets of subroutines that may be shared by many programs. The linker provides links from your executable file to these subroutines. Libraries can be of various types:

Library type	*Use*
Language-specific	Contain routines called by programs written in a specific language
PRIMOS standard	Contain routines called by programs in all languages
Application-specific	Contain routines used by some applications
User supplied	Contain routines created by users with BIND

In order to link your program successfully, you need to link all required libraries. Required language-specific libraries are listed in Table 11-1. Consult the documentation for the language you are using for more details.

Application and user libraries are not discussed in this book. For information on applications libraries see *Subroutines Reference IV: Libraries and I/O*. For information on user libraries, consult the *Programmer's Guide to BIND and EPFs* and the *Advanced Programmer's Guide, Vol I.*

TABLE 11-1
Language Specific Library Names

Compiler	*Library Name*
CC	C_LIB
CBL	CBLLIB
PASCAL	PASLIB
PL1	PLILIB
VRPG	VRPGLB

The following sections on BIND and LOAD explain how to link libraries.

Using BIND

BIND is the preferred linker for most object files. Invoke BIND with the command

BIND [*pathname*] [*options*]

The optional *pathname* identifies the EPF to be created. The *options* are BIND subcommands, the most important of which are explained in the next section.

You can use BIND in two ways:

- On a single command line
- As an interactive subsystem

Using a Single Command Line

When you use BIND with a single command line, the command line must include all *options*. The options can be any of the BIND subcommands discussed in the following sections. Each option must be preceded by a hyphen.

Using BIND Interactively

If *options* are not specified on the command line, the linking session is interactive, whether or not *pathname* is specified. In interactive mode, BIND prompts you for each subcommand with a colon (`:`). After executing a subcommand successfully, BIND repeats the `:` prompt.

If an error occurs during an operation in interactive mode, BIND displays an error message and then the prompt character. For a complete list of BIND error messages, see the *Programmer's Guide to BIND and EPFs*. When BIND encounters a system error, it displays a system error message, such as `File in use`, `Illegal name`, or `Insufficient access rights`, and returns the prompt symbol.

In interactive mode, BIND remains in control until either a FILE or a QUIT subcommand returns control to PRIMOS.

BIND subcommands and comment lines can be used in command files.

BIND Subcommands

Linking with BIND normally requires only a few straightforward commands. (BIND has many additional features that enable you to perform difficult linking tasks. These are described in the *Programmer's Guide to BIND and EPFs*.) The following commands accomplish most linking functions. Enter each command either as an option (preceded by a hyphen) on the command line or as a subcommand after the `:` (colon) prompt.

Command	*Function*
LOAD ***pathname_1*** **[... *pathname_n*]**	Links the object file(s) of one or more programs that have been compiled in V or I mode. You can omit the .BIN suffix, because BIND automatically looks for a filename with the .BIN suffix.
LIBRARY **[*libraryname_1* ... *libraryname_n*]**	Links one or more library files. If the library is supplied by Prime, it is located in the top-level directory LIB, and the library's objectname alone is adequate. If the library is user-supplied and not in LIB, a pathname is required. The objectnames for language-specific libraries are given in the Prime documentation for each language and in Table 11-1. Typing the command LIBRARY without any arguments links the standard system library, named PFTNLB, by default. All languages require that you link this library.

MAP [*option*]

Displays a memory map. The –UNDEFINED option lists unresolved references (usually either subroutines that have not been linked or misspelled references in the source code). The –FULL option gives a complete map. Mapping is explained in the *Programmer's Guide to BIND and EPFs.*

FILE [*pathname*]

Saves the newly created EPF as a file, with the filename suffix .RUN. You can determine the filename of the EPF in three ways:

- If you give a *pathname* argument to the FILE subcommand, the EPF is saved, using this pathname, as *pathname*.RUN.
- If you gave a *pathname* argument to the BIND command (but not to the FILE subcommand), the EPF is saved, using this pathname, as *pathname*.RUN.
- Otherwise, BIND uses the first pathname given to a LOAD subcommand to create *pathname*.RUN for the EPF.

When you use BIND interactively, you must end your session with the FILE subcommand in order to save the newly created EPF. If you use BIND in command line mode, the EPF is saved automatically, and the FILE subcommand is optional; you need to use it only if you want to specify a new name for the EPF.

After you save the EPF with the FILE subcommand, BIND returns you to PRIMOS command level.

QUIT

Returns you to PRIMOS command level without saving the current EPF.

$\textbf{HELP}\left[\begin{Bmatrix}\textit{\textbf{subcommand}}\\ \textbf{-LIST}\end{Bmatrix}\right]$

Displays an explanation and correct syntax of the specified BIND *subcommand.* The –LIST option lists all BIND subcommands.

Normal Linking

Use the following interactive procedure to link most programs:

1. Give the command BIND [*pathname*].
2. Use the LOAD subcommand to link the object file of your program and any separately compiled subroutines. If you are linking more than one file, you can either give the LOAD subcommand with several pathnames, or give a separate LOAD subcommand for each file.
3. Use the LIBRARY subcommand to link the required libraries. Remember that the default standard system library is always required. If you are linking several libraries, you can either give the LIBRARY subcommand once with several pathnames (or objectnames), or give a separate LIBRARY subcommand for each library.
4. At this point you should receive the `BIND COMPLETE` message. If you do not receive a `BIND COMPLETE` message, you have probably omitted or mispelled some library name during linking. If this is the case, use either the LOAD or the LIBRARY subcommand to relink from the point at which the omission or misspelling occurred.

If you have misspelled subroutine names in your program, you need to edit, recompile, and then relink the program. If you cannot easily identify the unresolved references, use the –UNDEFINED option of the MAP subcommand to help you. The MAP subcommand is discussed in the *Programmer's Guide to BIND and EPFs.*

5. When you have received the `BIND COMPLETE` message, use the FILE subcommand to save the EPF.

Remember that you have three ways to name your EPF:

- Giving the BIND command a pathname.
- Giving the FILE subcommand a pathname. This method of naming overrides any other.
- If you do neither of the above, BIND uses the name of the first object file linked with the LOAD subcommand.

BIND automatically adds the .RUN suffix.

Order of Linking

The following order of linking is recommended:

1. Main program (LOAD *pathname*).
2. Separately compiled user-supplied subroutines in the order in which they are called (LOAD *pathname*).
3. User-supplied libraries (LIBRARY *pathname*). BIND looks for them in the location specified in *pathname.*
4. Language-specific libraries (LIBRARY *libraryname*). See Table 11-1 for the names of the required language-specific libraries. Language-specific libraries are not required for programs written in FORTRAN IV, FORTRAN 77, and PMA.
5. Other Prime Libraries (LIBRARY *libraryname*), such as VAPPLB for the V-mode applications library, VSRTLI for the V-mode sort library, and MPLUSLB for the MIDASPLUS library.
6. Standard system library (LIBRARY), required by BIND to link programs in all languages.

BIND Examples

The following examples assume that your current directory contains an object file named CIRCLE.BIN. The F77 compiler created this object file when it compiled your program CIRCLE.F77. In the first example, the BIND command itself is used to name the EPF:

```
OK, BIND CIRCLE
[BIND Rev 22.0 Copyright (C) 1987, Prime Compter, Inc.]
: LOAD CIRCLE
: LIBRARY
BIND COMPLETE
: FILE
OK,
```

Because the LOAD subcommand recognizes the .BIN suffix for object files, the suffix need not be typed. Also, because this program is written in FORTRAN 77, only the standard system library needs to be linked, so the LIBRARY subcommand is used only once, without a pathname argument. (Remember, the standard system library is the default value.)

Noninteractively, the same linking session can be accomplished on a single command line:

```
OK, BIND CIRCLE -LOAD CIRCLE -LIBRARY
[BIND Rev 22.0 Copyright (C) 1987, Prime Compter, Inc.]
BIND COMPLETE
OK,
```

When BIND is used noninteractively, the FILE subcommand is added by default to the end of the command line and need not be typed.

Both of the above examples can be simplified, however. Because the EPF uses the name of the first object file linked (that is, CIRCLE), the BIND command needs no argument:

```
OK, BIND
[BIND Rev 22.0 Copyright (C) 1987, Prime Compter, Inc.]
: LOAD CIRCLE
: LIBRARY
BIND COMPLETE
: FILE
OK,
```

The noninteractive version is

```
OK, BIND -LOAD CIRCLE -LIBRARY
[BIND Rev 22.0 Copyright (C) 1987, Prime Compter, Inc.]
BIND COMPLETE
OK,
```

In all cases, BIND saves the EPF CIRCLE.RUN in your current directory.

If you do not give the BIND command an argument and do not want the EPF to be named after the first object file linked, you must use the FILE subcommand to name your EPF:

```
OK, BIND
[BIND Rev 22.0 Copyright (C) 1987, Prime Compter, Inc.]
: LOAD CIRCLE
: LIBRARY
BIND COMPLETE
: FILE AREA_CIRCLE
OK,
```

The noninteractive version uses the –FILE option:

```
OK, BIND -LOAD CIRCLE -LIBRARY -FILE AREA_CIRCLE
[BIND Rev 22.0 Copyright (C) 1987, Prime Compter, Inc.]
BIND COMPLETE
OK,
```

In both cases, BIND saves the EPF AREA_CIRCLE.RUN in your current directory.

Finally, here is an example of retrying a linking session when you do not receive the BIND COMPLETE message. In this case, the main Pascal program calls a PL/I subprogram.

```
OK, BIND
[BIND Rev 22.0 Copyright (C) 1987, Prime Compter, Inc.]
: LOAD PASS                                    Link the main program.
: LOAD VAL                                   Link the called program.
: LIBRARY PASLIB                              Link the Pascal library.
: LIBRARY PL1GLB                               Link the PL1G library.
                                      No BIND COMPLETE; message,
                                        you forgot the standard library.

: LIBRARY                                  Link the missing standard
                                                     system library.
BIND COMPLETE
: FILE                                       Save the EPF and return
OK,                                                  to PRIMOS level.
```

After a successful linking session, you can execute the EPF at PRIMOS command level with the RESUME command, as detailed in Chapter 12.

Using LOAD

Use the LOAD utility for linking object files compiled in 32R or 64R mode. LOAD creates executable runfiles identified by the .SAVE suffix.

Note

The following description emphasizes the linker commands and functions that are of most use to the FORTRAN programmer. For a complete description of all linker commands, including those for advanced system-level programming, see the *SEG and LOAD Reference Guide*.

To invoke the LOAD linker, use the command

```
LOAD
```

This transfers control to the R-mode linker, which displays a dollar-sign ($) prompt and awaits a LOAD subcommand. After executing a command successfully, LOAD repeats the $ prompt.

If an error occurs during an operation, LOAD displays an error message and then the $ prompt. For a complete list of LOAD error messages, see the *SEG and LOAD Reference Guide*. When LOAD encounters a system error it displays the system error message and then the $ prompt.

LOAD remains in control either until a QUIT or PAUSE subcommand returns control to PRIMOS or an EXECUTE subcommand starts execution of the linked program.

You can use LOAD subcommands in command files.

Normal Linking

Using LOAD normally requires only a few straightforward commands. LOAD has many additional features that enable you to perform difficult linking tasks. These are described in the *SEG and LOAD Reference Guide.*

The following commands accomplish most linking functions.

Helpful PRIMOS Commands:

Command	*Function*
FILMEM	Initializes user space in preparation for linking.
LOAD	Invokes LOAD for entry of subcommands.

LOAD Subcommands:

Command	*Function*
DC	Defers linking of COMMON until everything else has been linked. This delay prevents overlap of COMMON and program areas.
MODE ***option***	Sets runfile addressing mode as 32R (default) or 64R.
LOAD ***pathname***	Links specified object file. The .BIN need not be typed.
LIBRARY [***libraryname***]	Links specified library object file from the top-level directory LIB. (Default is FTNLIB.)
MAP [*option*]	Displays a memory map. The options are numbered. MAP –3 shows unresolved references.
INITIALIZE	Returns LOAD to starting condition in case of command errors or faulty linking.
SAVE [*pathname*]	Saves the runfile *pathname*.SAVE in the specified directory. If *pathname* is not given, LOAD uses the name of the first object file linked and adds the .SAVE suffix.
QUIT	Returns you to PRIMOS command level.

Most linking sessions can be accomplished with the following basic procedures:

1. Use the PRIMOS command FILMEM to initialize memory.
2. Invoke LOAD.
3. Use the MODE subcommand to set the addressing mode, if necessary. (The default is 32R mode.)
4. Use the LOAD subcommand to link the object file and any separately compiled subroutines. (LOAD searches first for filenames plus the .BIN suffix, then for the given filenames without the suffix.)
5. Use the LIBRARY subcommand to link subroutines called from libraries. The default library is FTNLIB in the top-level directory LIB. Other libraries, such as SRTLIB or APPLIB, must be named explicitly. (Link user supplied libraries with the LOAD subcommand.)

6. If you do not receive a `LOAD COMPLETE` message, use MAP –3 to identify the unsatisfied references, and link them. (If you use the DC option, the `LOAD COMPLETE` message may not appear until you give the SAVE command.)
7. SAVE the runfile, either by giving an appropriate name or by allowing LOAD to create a default filename.
8. Use the QUIT subcommand to return to PRIMOS.

If Step 6 produces a `LOAD COMPLETE` message, then linking was accomplished. Any problem becomes apparent by the absence of a `LOAD COMPLETE` message or some other LOAD error message.

After a successful linking session, you can execute your runfile from PRIMOS command level with the RESUME command. See Chapter 12.

The following example assumes that the source program TEST.FTN was compiled in 32R mode to produce the object file TEST.BIN.

```
OK, FILMEM
OK, LOAD
$ DC
$ LOAD TEST
$ LIBRARY
LOAD COMPLETE
$ SAVE
$ QUIT
OK,
```

The user's current directory now contains the runfile TEST.SAVE, ready to be executed by the RESUME command.

Order of Linking

The order of linking and procedures for mapping are the same for LOAD as they are for BIND. Note, however, that you use the LOAD subcommand instead of the LIBRARY subcommand to link user-supplied libraries with LOAD.

12 Running Programs Interactively

PRIMOS provides three environments for executing programs:

- Interactive
- Phantom
- Batch

When you execute a program interactively, your terminal is **dedicated** to the program during execution. You may halt the program or give input from the keyboard, program output is displayed on the screen, and any error messages are directed to the screen. While the program is running, you can't use the terminal for anything else.

Programs executed as phantoms or in the Batch environment don't communicate directly with your terminal. Any program that does not require user input can be executed in this way in order to free up your terminal for other work while the program is executing. Phantoms and the Batch environment are discussed in Chapter 16.

This chapter shows you how to run programs interactively. Topics covered include

- Using the interactive environment
- Running programs with the RESUME command
- Restarting interrupted programs with the START command
- Error messages
- The command environment

Using the Interactive Environment

Typical uses for interactive execution are

- Program development and debugging
- Programs requiring short execution time

- Data entry programs
- Interactive programs, such as ED

Most of the time, you run programs interactively because they require some input at the terminal or produce output on the screen. However, you can interactively run a program that requires no terminal input and produces no screen output. Your terminal is still dedicated to such a program while it runs; you cannot run other programs until the program terminates and you receive the PRIMOS `OK,` prompt. Short programs and programs being tested and debugged are often run this way even if they do not require user input or produce screen output.

Invoking Programs to Execute Interactively

You can invoke programs to execute interactively in a variety of ways:

- From the command line
- From a command input file
- From a CPL program

The three possibilities mean that programs that run interactively don't necessarily have to be invoked interactively from the terminal. In the first case, you invoke a program interactively by giving a command at the terminal. In the other cases, your command input file or CPL program invokes the program. Nevertheless, programs invoked from command files and CPL programs can run interactively. If you always begin your terminal sessions by working with EMACS, for example, you can have your login.CPL invoke EMACS for you.

You invoke programs to run interactively with the same commands, whether you do so from the command line or from a command input file or CPL program.

The RESUME Command

You execute EPFs, R-mode static runfiles, and CPL programs using the RESUME command.

The format is

RESUME ***pathname***

pathname is the name of the program to be executed interactively. If the program is in the current directory, you can use the filename alone.

Note

Use the SEG command to execute static V-mode or I-mode runfiles that were linked with the SEG utility. For more information on the SEG command, see the *PRIMOS Commands Reference Guide*.

Runfile Naming Conventions

In most cases, a program must have a name ending in one of the the standard PRIMOS runfile suffixes: .RUN for EPFs, .SAVE for R-mode programs, and .CPL for CPL programs. The only exceptions to this rule are R-mode runfile names. An R-mode runfile may have a name without any suffix. Other types of runfiles must use one of the standard suffixes. If they do not, RESUME attempts to execute them as R-mode files, resulting in an error.

When you give the RESUME command, you can give *pathname* without using the suffix, if you wish. RESUME searches for a file with one of the correct suffixes in the following order:

Pathname	*Assumption*
***pathname*.RUN**	File is an EPF.
***pathname*.SAVE**	File is an R-mode runfile.
***pathname*.CPL**	File is a CPL program.
pathname	File is an R-mode runfile.

For example, if you type

```
OK, RESUME CIRCLE
```

RESUME looks in your current directory first for CIRCLE.RUN, then for CIRCLE.SAVE, then for CIRCLE.CPL, and last for CIRCLE. RESUME executes any file with an unsuffixed name as a .SAVE runfile. In cases where ambiguity is possible, give the suffix.

After the RESUME command has completed executing a program, or if an error is encountered, control returns to PRIMOS command level, and you receive the appropriate screen prompt.

The START Command

You can interrupt a program by typing Ctrl P at the terminal. Programs can also be interrupted by a runtime error or, in the case of FORTRAN programs, by a PAUSE statement in the program. You can often restart an interrupted program using the START command.

The format is

START

This command attempts to restart the execution of a suspended program at the point where it was interrupted. For example,

```
OK, RESUME CIRCLE
 TYPE IN A VALUE FOR THE RADIUS OF THE CIRCLE:

Ctrl P                                          Suspends the program.
```

```
QUIT.
OK, START
33
 THE AREA OF THE CIRCLE IS:    207.63
OK,
```

START does not always succeed in restarting suspended programs. Programs that are interrupted by errors may not be able to execute further. Your ability to restart a program may also depend on the number of programs you have suspended simultaneously. The START command treats static-mode and dynamic-mode programs differently.

Restarting Static-mode Programs

If you suspend a static-mode program and then invoke another static-mode program, the second overwrites the first. Therefore, you can only restart the most recently invoked static-mode program.

Restarting EPFs

You can suspend and restart EPFs much more freely than static-mode programs. PRIMOS can bring EPFs into any free area of memory, so that several EPFs can exist simultaneously without overwriting each other. This means that you can invoke an EPF, suspend it, invoke other programs, and then restart the first EPF without interference from other programs.

For example, you can execute two EPFs called CIRCLE.RUN and DEGREE.RUN in the following way:

```
OK, RESUME CIRCLE
 TYPE IN A VALUE FOR THE RADIUS OF THE CIRCLE:

Ctrl P                                             You interrupt the program.

QUIT.
OK, RESUME DEGREES
 ENTER A TEMPERATURE IN FAHRENHEIT DEGREES:
32
 THE CONVERSION TO CENTIGRADE IS:     0.000000
OK, START                                   You restart the first program and
33                                          input the value requested earlier.
 THE AREA OF THE CIRCLE IS:    207.631
OK,
```

You cannot carry out the same procedure with two static-mode programs. If the two runfiles are static-mode programs called CIRCLE.SAVE and DEGREES.SAVE, for example, the START command returns the following error message:

```
Attempt to proceed to overwritten program image. (listen_)
ER!
```

because DEGREES.SAVE has overwritten CIRCLE.SAVE. To rerun CIRCLE.SAVE in this case, you have to use the RESUME command and execute the program from the beginning again.

Using START With Several EPFs

You can successively suspend and restart several EPFs. Successive START commands attempt to restart the programs beginning with the one most recently suspended. For example,

```
OK, RESUME CIRCLE
 TYPE IN A VALUE FOR THE RADIUS OF THE CIRCLE:

Ctrl P

QUIT.
OK, RESUME DEGREES
 ENTER A TEMPERATURE IN FAHRENHEIT DEGREES:

Ctrl P

QUIT.

OK, START
32
 THE CONVERSION TO CENTIGRADE IS:     0.000000

OK, START
33
 THE AREA OF THE CIRCLE IS:    207.631
OK,
```

Your System Administrator sets the number of suspended EPFs that you can maintain and then restart with the START command. This limit is discussed below in the section, The Command Environment.

Runtime Error Messages

When you execute a program with RESUME or restart one with START, PRIMOS may detect error conditions. If the program cannot continue execution, it is suspended, a runtime error message is generated, and you are returned to PRIMOS command level and receive an error prompt. Appendix D gives a list of runtime error messages.

The Command Environment

Whenever you invoke a program, you make use of a section of PRIMOS called the command environment. The **command environment** is the part of PRIMOS that is responsible for interpreting commands typed at the terminal, invoking and suspending the requested programs, and displaying prompts. Among its other tasks, the command environment records and maintains information about the status and operation of each program you invoke. When you restart a suspended EPF, the command environment uses the information it has stored to get the program running from the point at which it stopped.

Command Environment Limits

The command environment lets you work with EPFs in a flexible manner. You can invoke an EPF, suspend its operation, issue commands or invoke other programs, and then restart the suspended EPF at the point where it stopped. In fact, you can have several EPFs suspended at once and still be able to restart all of them. You can do all this because PRIMOS can maintain several suspended EPFs in memory at once without conflict. However, doing so occupies system resources, and this makes some constraints necessary. Programs take up room in memory, for example. While virtual memory gives you a large memory space, it is not an unlimited space. Moreover, your virtual memory space must share the computer's physical resources with other users. Limits on individual use are necessary in order to maintain system performance for all users.

These limits, called **command environment limits**, are set for you by the System Aministrator when your login ID is established. This section describes command environment limits and shows you how to deal with situations in which you have reached the limits established for your user ID.

The System or Project Administrator establishes four limits for each user ID:

- Command environment depth
- Command environment breadth
- Number of dynamic segments that can be allocated in your private address space
- Number of static segments that can be allocated in your private address space

Command Environment Depth: Whenever a program is suspended (either because of an error or because you type Ctrl P) the command environment returns you to PRIMOS command level. Each time this happens, you are said to be at a *new* command level, because the command environment maintains the information that it needs if you later attempt to restart the suspended program. By putting you at a new command level, the command environment allows you to go back later and try to take care of any unfinished business from previous command levels.

Note that this is quite different from what happens when a program runs to completion. In that case, the command environment no longer needs to keep the program in a state of suspended animation while you carry out other work. Therefore, it keeps you at the same command level. The maximum number of command levels allowed for your user ID is called the **command environment depth**. When you log in, you are at command level 1. Every time a program is suspended, you enter a new command level. If the number of command levels reaches the command environment depth, you cannot invoke any more programs.

Command Environment Breadth: One program can invoke another without returning you to PRIMOS command level and without increasing your command level. The maximum number of such invocations allowed in one command level is called the **command environment breadth**.

Segment Limits: When you invoke a program, PRIMOS allocates one or more segments in your address space to the program. PRIMOS allocates either static or dynamic segments, depending on the type of program to be run.

Static-mode programs created by LOAD and SEG are loaded into predefined static segments. A static-mode program overwrites any other static-mode program previously invoked.

Programs created by BIND are called **dynamic** because they are mapped at runtime into any unused dynamic segment or segments. They can coexist with other dynamic programs, because one dynamic mode program does not overwrite another.

The System Administrator sets limits on the maximum numbers of dynamic and static segments that can be allocated within your private address space. You cannot invoke more programs than can fit in the segments allotted to your private address space.

Checking Your Limits: To find out what command environment limits the Project or System Administrator has set for your user ID, use the PRIMOS LIST_LIMITS command. The format is

```
LIST_LIMITS
```

LIST_LIMITS displays

- The number of command levels you can use (command environment depth)
- The number of programs you can invoke at any command level (command environment breadth)
- The number of private dynamic segments you can use
- The number of private static segments you can use

The following example shows how LIST_LIMITS displays this information.

```
OK, LIST_LIMITS

Maximum number of command levels: 10
Maximum number of program invocations: 10
Maximum number of private static segments: 100
Maximum number of private dynamic segments: 150

OK,
```

Exceeding Limits: Segment and program invocation limits normally concern only the advanced programmer. For more information, refer to the chapter on troubleshooting in the *Programmer's Guide to BIND and EPFs*.

Command environment depth concerns anyone using PRIMOS who types Ctrl P once too often. If you invoke and suspend several EPFs, it is possible to reach the maximum number of command levels allowed. The rest of this section shows you what to do when you exceed the maximum number of command levels for your user ID.

Mini-command Level

If you exceed the number of command levels allotted to you, PRIMOS does not allow you to go on invoking programs or giving other PRIMOS commands normally. Instead, PRIMOS places you at a special command level called **mini-command level**. When you are at mini-command level, you are allowed to give only a very limited subset of PRIMOS commands. These commands are intended to help you diagnose the situation and correct it by reducing your command level and freeing up system resources.

The following example shows the display you see when you reach mini-command level.

```
You have exceeded your maximum number of command levels.

You are now at mini-command level.  Only the commands shown
below are available.  Of these, RLS -ALL should return you to
command level 1.  If it does not, type ICE.  If this problem
recurs, contact your System Administrator.


Valid mini-commands are:

Abbrev  Full name                       Abbrev  Full name
------  ------------------------------  ------  ------------------------------
C       CLOSE                           COMO    COMOUTPUT
DMSTK   DUMP_STACK                      ICE     INITIALIZE_COMMAND_ENVIRONMENT
LE      LIST_EPF                        LL      LIST_LIMITS
LMC     LIST_MINI_COMMANDS              LS      LIST_SEGMENT
        LOGIN                           LO      LOGOUT
P       PM                              PR      PRERR
        RDY                             REN     REENTER
RLS     RELEASE_LEVEL                   REMEPF  REMOVE_EPF
S       START
```

At mini-command level, you can execute only those commands listed in the display above. Using any other command, including personal abbreviations, results in an error message:

```
OK, LD
Invalid command "LD" - only mini level commands accepted. (min$cp)
ER!
```

Typing another Ctrl P produces this message:

```
Terminal QUIT invalid now.  (listen_)
```

Mini-commands

Table 12-1 lists a few of the most useful mini-commands. The remainder of this section shows you how to use these commands to escape from mini-command level.

Note

Other mini-commands, such as COMOUTPUT and LOGOUT, are documented elsewhere in this book. Mini-commands for more advanced operations are documented in the *PRIMOS Commands Reference Guide* and the *Programmer's Guide to BIND and EPFs.*

TABLE 12-1
Useful Mini-commands

Command	*Action*
LIST_LIMITS	Determines your command environment limits
RDY –LONG	Monitors command level
RELEASE_LEVEL	Releases unneeded command levels
INITIALIZE_COMMAND_ENVIRONMENT	Reinitializes command environment
LIST_EPF	Determines the status of EPFs
LIST_MINI_COMMANDS	Determines which commands are available

RDY –LONG and RELEASE_LEVEL: RDY –LONG and RELEASE_LEVEL are useful whenever you type Ctrl P, even if you are not using an EPF. Whenever your command level is greater than one, PRIMOS displays your command level as the last element of the long prompt. Thus, invoking long prompts with RDY –LONG is a convenient way to monitor your current command level as you work.

For example,

```
OK, RDY -LONG
OK 17:26:38  3.166  2.236
Ctrl P
QUIT.
OK 17:26:42  0.045  0.190  level 2
```

Pressing Ctrl P moves you up one command level. Using the RELEASE_LEVEL command moves you down one command level:

```
OK, RDY -LONG
OK 17:26:38  3.166  2.236
Ctrl P
QUIT.
OK 17:26:42  0.045  0.190  level 2
Ctrl P
QUIT.
OK 17:26:44  0.033  0.000  level 3
Ctrl P
QUIT.
```

```
OK 17:26:45  0.036  0.000  level 4
Ctrl P
QUIT.
Now at command level 5. To release use RLS. (listen_)
OK 17:26:48  0.057  0.000  level 5
RLS
OK 17:26:54  0.039  0.000  level 4
RLS
OK 17:26:58  0.045  0.000  level 3
RLS
OK 17:27:01  0.039  0.000  level 2
RLS
OK 17:27:04  0.042  0.000
```

By default, RELEASE_LEVEL moves you down one command level. But when you specify the –ALL option, RELEASE_LEVEL brings you back to command level 1, regardless of the level from which you start:

```
Ctrl P
QUIT.
Now at command level 5. To release use RLS. (listen_)
OK 17:27:30  0.054  0.000  level 5
RLS -ALL
OK 17:27:45  0.078  0.151
```

As the above examples show, PRIMOS prompts you to use the RELEASE_LEVEL command when you reach command level 5. PRIMOS prompts you again to use RELEASE_LEVEL after every additional five levels. If you obey the system's prompts and monitor your current command level with RDY –LONG, you should never reach mini-command level. If you do reach mini-command level, the easiest way to escape is to use RELEASE_LEVEL –ALL.

Bear in mind, however, that if you interrupt a program with Ctrl P, and then use RELEASE_LEVEL, you cannot use the START command to restart the program from the point of interruption. Whenever you release a command level, PRIMOS no longer maintains the information necessary to restart the program invoked at that level. If you use RELEASE_LEVEL –ALL, you can't restart any of the suspended programs.

INITIALIZE_COMMAND_ENVIRONMENT: The most drastic way to escape from mini-command level is to use INITIALIZE_COMMAND_ENVIRONMENT. Its effect is similar to logging out and then logging in again. Use this command only if RELEASE_LEVEL –ALL fails to free you from mini-command level.

INITIALIZE_COMMAND_ENVIRONMENT performs the following actions:

- Closes all open files, including any open COMOUTPUT files
- Deallocates all your private dynamic and static segments
- Resets your command environment to an initial state
- Resets erase and kill characters and other terminal characteristics to system defaults

- Executes your login file

After you execute this command, the following conditions exist:

- You are attached to your initial attach point.
- You are unable to restart any suspended or active EPFs because all EPFs have been unmapped from your address space.
- Any COMO file recording your terminal session has been terminated.

LIST_EPF: LIST_EPF displays information about EPFs available to you. It is especially useful when you have escaped from mini-command level and want to find out if an EPF can still be restarted.

The format of LIST_EPF is

LIST_EPF [*EPFname-1* [... *EPFname-8*]] [*options*]

If you specify LIST_EPF without arguments or options, PRIMOS displays information about the EPFs currently mapped to your address space. An EPF is **mapped** if it has been allocated room in your memory space. When you invoke an EPF for the first time during a terminal session, PRIMOS maps it to your memory space. It remains mapped for some time during the terminal session so that it can be reinvoked rapidly.

You can specify up to eight EPF names for a display of information on specific EPFs. You can also use wildcards to select several EPFs with similar names. If you specify EPF names, you need only supply the objectnames, and you can omit the .RUN suffix.

Note

An EPF is **unmapped** if it is available in your file system, but is not mapped to your memory space. You can specify the –NOT_MAPPED option for a listing of EPFs not mapped to your address space. In this case you need to specify pathnames for EPFs not in your current directory.

LIST_EPF lists program EPFs and library EPFs separately. Program EPFs are programs that you can invoke from the command line, including many PRIMOS commands. (Library EPFs contain routines that are called by other programs.) You can examine the status of your program EPFs to determine whether they can be restarted.

The status of a mapped EPF can be either not active or active. A program EPF is **not active** when it has been invoked and run successfully to conclusion. It remains mapped to your address space so that it can be quickly reinvoked and run from the beginning. A program EPF is **active** when it has been invoked and then suspended. You can often restart an active EPF from the point at which it stopped using the START command.

The following example shows the LIST_EPF display after you have invoked the programs DEGREES.RUN, CIRCLE.RUN, and TEST.RUN. You have let CIRCLE.RUN and TEST.RUN run to completion, but you have interrupted the execution of DEGREES.RUN with Ctrl P.

```
OK, LIST_EPF
```

```
1 Process-Class Library EPF.

(active)     <SYS001>LIBRARIES*>SYSTEM_LIBRARY.RUN


2 Program-Class Library EPFs.

(not active) <SYS001>LIBRARIES*>APPLICATION_LIBRARY.RUN
(active)     <SYS001>LIBRARIES*>FORTRAN_IO_LIBRARY.RUN


3 Program EPFs.

(not active) <DISK01>USER>PROGRAMS>CIRCLE.RUN
(active)     <DISK01>USER>PROGRAMS>DEGREES.RUN
(not active) <DISK01>USER>PROGRAMS>TEST.RUN

OK,
```

The display shows DEGREES.RUN as `active`. You may be able to restart DEGREES.RUN with the START command.

Notes

If you have more than one active EPF, the LIST_EPF display does not tell you in what order the EPFs were suspended or will be restarted by successive START commands.

When you use RELEASE_LEVEL to release a command level, any program EPF associated with that level is unmapped and no longer appears in the listing of mapped EPFs.

LIST_MINI_COMMANDS: If you forget what commands you can use at mini-command level, use LIST_MINI_COMMANDS to display them. The command's format is

LIST_MINI_COMMANDS [*command_match*]

The optional argument *command_match* is a character string, possibly containing wildcards, that is used as a pattern match for the commands allowed at mini-command level. If you omit this argument, the names of all the commands allowed at mini-command level are listed alphabetically.

The following example shows the display you get when you specify LIST@@ as a command match argument:

```
OK, LIST_MINI_COMMANDS LIST@@
Abbrev   Full name                         Abbrev   Full name
-----------------------------------------------------------

LE       LIST_EPF                          LL       LIST_LIMITS
LMC      LIST_MINI_COMMANDS                LS       LIST_SEGMENT

OK,
```

13 Debugging Programs

Most programs don't work as expected the first time you run them. Developing programs often requires repeated testing and modification of the source code. To make this process easier, Prime offers a Source Level Debugger (DBG) as a separately priced product.

This chapter gives a brief working introduction to DBG. It covers the following topics:

- DBG code requirements
- The features of DBG
- Compiling and linking programs for debugging with DBG
- Invoking and terminating DBG
- Basic DBG subcommands to examine source code, start and stop execution, and modify data
- A sample debugging session

DBG Code Requirements

DBG lets you debug programs written in six Prime supported high-level languages:

- C
- FORTRAN IV (V mode only)
- FORTRAN 77
- Pascal
- PL/I
- RPG II (V Mode)

As the list above suggests, you can use DBG to debug programs compiled in 64V or 32I mode only. You cannot use DBG with R-mode programs.

DBG debugs both EPFs and static V-mode and I-mode code (.SEG runfiles) in the same way. You create and edit your program with one of the Prime text editors, compile it with the –DEBUG option, link it as usual, then execute and test it interactively under the control of DBG.

The Features of DBG

DBG lets you control, monitor, and manipulate the execution of your program. DBG's features are categorized as follows:

- Program control
- Data manipulation
- Tracing
- Miscellaneous

DBG is an interactive program. You carry out operations by giving DBG subcommands, and DBG shows the results on the screen. In the following sections, each feature is introduced along with the DBG subcommand that implements it.

Program Control Features

Program control features let you use DBG to control the execution of your program. For example, you can start or restart execution, suspend execution, execute one statement at a time, and call any procedure, function, or subroutine. Use these features to discover where and why your program failed.

Here is a partial list of DBG's program control features and related subcommands:

- Restarting the program. At any time during your debugging session, you can begin execution of the program being debugged, no matter where execution is suspended (RESTART subcommand).
- Setting breakpoints. You can suspend your program's execution at any executable statement or at any entry to or exit from a procedure, function, or subroutine (BREAKPOINT subcommand).
- Using breakpoint action lists. You can specify that one or more debugger commands be executed each time a breakpoint occurs (BREAKPOINT subcommand).
- Using conditional action lists. You can specify that a breakpoint action list be executed only if a given condition is true (BREAKPOINT and IF subcommands).
- Setting conditional breakpoints. You can specify that a breakpoint occur only if a given condition is true (BREAKPOINT subcommand).
- Continuing program execution. You can resume program execution after execution has been suspended (CONTINUE subcommand).
- Single stepping. You can execute one or more statements at a time; step across, into, and out of procedures, functions, or subroutines (STEP, STEPIN, IN, and OUT subcommands).
- Displaying breakpoints and tracepoints. You can display one or more breakpoints or tracepoints (LIST and LISTALL subcommands).
- Deleting breakpoints and tracepoints. You can delete one or more breakpoints or tracepoints (CLEAR and CLEARALL subcommands).
- Transferring control. You can transfer the position at which execution is to resume from one statement in your program to another (GOTO subcommand).

- Calling procedures, functions, and subroutines. From DBG command level, you can call a procedure, function, or subroutine, supplying argument lists if needed (CALL subcommand).

Data Manipulation Features

Another important part of debugging is the ability to examine, evaluate, and modify expressions. Whenever execution is suspended, you can examine and modify the values of variables and expressions and examine their data types.

Here is a partial list of DBG's data manipulation features and related subcommands:

- Examining variables. DBG can display the value of any scalar, array, or structured variable in different, selectable data formats (: subcommand).
- Evaluating expressions. You can evaluate any expression allowed by any source language (: subcommand).
- Assigning values. You can modify the value of a variable (LET subcommand).
- Examining data types. You can examine the data type of a variable or expression (TYPE subcommand).
- Examining the values of arguments. You can examine the values of arguments passed to procedures, functions, or subroutines (ARGUMENTS subcommand).
- Using built-in functions. You can use any built-in function supported by the source language to help evaluate a variable or an expression.

Tracing Features

Tracing features let you trace the progress of your program's execution from beginning to end. For example, you can have trace messages displayed at strategic points during program execution, and you can trace the value of a variable as it changes throughout program execution. Here is a partial list of DBG's tracing features and related subcommands:

- Setting tracepoints. You can specify that a trace message be displayed at the execution of a specified statement or at the entry to or exit from a procedure, function, or subroutine (TRACEPOINT subcommand).
- Tracing values. You can specify that a message be displayed whenever the value of a specified variable changes during execution. This message tells you the old value, the new value, and the location in your program where the change was detected (WATCH, VTRACE, UNWATCH, and WATCHLIST subcommands).
- Tracing statements. You can specify that a trace message be displayed prior to the execution of each statement and/or each labeled statement (STRACE subcommand).
- Tracing at entries and exits. You can specify that a message be displayed each time any procedure, function, or subroutine is called or returns (ETRACE subcommand).
- Tracing the currently active program blocks. Throughout your program's execution, DBG can display a list of the currently active program block calls by displaying the call/return stack (TRACEBACK subcommand).

Miscellaneous Features

Here is a partial list of DBG's miscellaneous features and related subcommands:

- Examining the source file. You can look at, but not change, your source files while debugging (SOURCE subcommand).
- Creating DBG command macros. You can create a macro to take the place of one or more DBG subcommands (MACRO and MACROLIST subcommands).
- Saving breakpoints, tracepoints, and macros. You can save all of your breakpoints, tracepoints, and macros in PRIMOS files, then use them again in future debugging sessions (SAVESTATE and LOADSTATE subcommands).
- Getting help. You can ask DBG for help in understanding subcommand syntax definitions (HELP subcommand).

More Information on DBG

This chapter offers an introduction to a few basic subcommands and techniques for debugging programs with DBG. To learn more about using DBG, consult the *Source Level Debugger User's Guide*. For a summary listing of DBG subcommands, see the *Loading and Debugging Companion.*

In addition, users with a knowledge of assembly language can use the Prime machine-level debuggers, PSD, VPSD, and ISPD. VPSD can be invoked from DBG to debug V-mode programs written in any language. PSD debugs R-mode programs and cannot be invoked from DBG. The *Assembly Language Programmer's Guide* contains a full reference text for these debuggers. The *Loading and Debugging Companion* contains a summary list of their subcommands.

Compiling and Linking the Program

To use DBG, you must first compile your code in either V or I mode using the compiler's –DEBUG option. For example, suppose that your current directory contains a FORTRAN IV program called MYPROGRAM.FTN. To debug MYPROGRAM with DBG, you must first compile the program using the –DEBUG option, as in the following example:

```
OK, FTN MYPROGRAM -DEBUG
```

Once you have successfully debugged your program, recompile it without the –DEBUG option in order to save file space. Compilation with the –DEBUG option produces object files and runfiles substantially larger than those produced by compilation without the –DEBUG option.

Link the program normally, as in the following examples:

```
OK, BIND
[BIND Rev 22.0 Copyright (C) 1987, Prime Computer, Inc.]
: LOAD MYPROGRAM
: LIBRARY
BIND COMPLETE
: FILE

OK, BIND -LOAD MYPROGRAM -LIBRARY
[BIND Rev 22.0 Copyright (C) 1987, Prime Computer, Inc.]
BIND COMPLETE
OK,
```

Invoking and Terminating the Debugger

Invoke the debugger from PRIMOS command level with the command

DBG ***pathname***

where *pathname* is the name of the EPF or .SEG runfile to be debugged. If the program is in the current directory, you can use the filename alone. DBG recognizes the .RUN and .SEG runfile suffixes (in that order); you need not type them on the command line. For example, to debug a program in the current directory called MYPROGRAM.RUN, enter

```
OK, DBG MYPROGRAM
```

DBG reads the program and symbol table for the EPF into memory and displays an identification message. When the the debugger's prompt character, a right angle bracket (>) appears at the left margin of the screen, DBG is ready to accept subcommands from the terminal.

Ending a DBG session: To terminate the debugging session and return to PRIMOS command level, give the DBG subcommand

```
> QUIT
```

Examining the Source Code

During debugging, you often need to examine you program's source code. You do this with the SOURCE subcommand. The format of the SOURCE subcommand is

SOURCE ***argument***

Most SOURCE arguments are the same as ED subcommands. They have the same functions as the corresponding ED subcommands used to locate and display text. (You cannot use

SOURCE to modify your source code.) As with ED, SOURCE maintains and manipulates a pointer to one line of your source code. For example, if you give DBG the command

```
> SOURCE POINT 4
```

DBG moves the source pointer to line 4 of your file and displays it on the screen, just as the ED command POINT 4 does during an ED session. Table 13-1 lists those SOURCE arguments that are taken from ED. For the other arguments to the SOURCE subcommand, see the *Source Level Debugger User's Guide*.

TABLE 13-1
SOURCE Subcommand Arguments

Argument	*Function*
TOP	Positions the line pointer to the top of the file
BOTTOM	Positions the line pointer to the bottom of the file
BRIEF	SOURCE doesn't display target lines of FIND, LOCATE, POINT, and NEXT operations
VERIFY	SOURCE displays target lines of FIND, LOCATE, POINT, and NEXT operations
PRINT	Displays line(s)
WHERE	Displays the current line number
POINT	Positions the line pointer to a specific line
NEXT	Moves the line pointer forward or backward
MODE	Sets edit mode; the only modes implemented are NUMBER/NNUMBER
LOCATE	Locates the line containing the specified text string
FIND	Locates the line with the specified text string beginning in a specified column
PSYMBOL	Displays a list of character symbols
SYMBOL	Sets the specified character symbol

The following example shows output from the SOURCE subcommand.

```
> SOURCE PRINT 30
    1:         J = 0
    2:         K = 0
    3:         DO 30 I = 1,10
    4:         J = J + 1
    5:         K = K + 1
    6:         CALL TEST (J,K)
    7:         WRITE (1,20) J,K
    8: 20      FORMAT (2I5)
    9: 30      CONTINUE
   10:         CALL EXIT
   11:         END
   12: C
   13: C
   14:         SUBROUTINE TEST (J,K)
   15: 10      J = J + 1
   16: 20      K = K + 1
   17:         IF (J .EQ. 5) GO TO 100
   18:         IF (J .EQ. 6) GO TO 110
   19: 100     IF (K .EQ. 10) GO TO 200
   20: 110     IF (K .EQ. 20) GO TO 210
   21:         GO TO 20
   22: 200     J = J + K
   23:         GO TO 300
   24: 210     J = K
   25: 300     RETURN
   26:         END
 BOTTOM
```

SOURCE Line Numbers

SOURCE displays your source code with line numbers. As the previous example shows, these numbers may not correspond to any statement numbers included in your program. Many DBG subcommands take arguments that indicate a line of source code. You can use the line numbers displayed by the SOURCE subcommand as arguments to such DBG subcommands.

Note

The discussion in this chapter deals with programs that consist of a single source file. When a program source consists of several files, such as a main program and separately compiled subroutines, SOURCE treats each source file separately. When you enter DBG, the file listed by SOURCE is the one containing the main program block. As you debug the program, stopping execution at different points, the file listed by SOURCE is generally the one that includes the program block where execution stopped. For detailed information about how the current source file is determined, see the *Source Level Debugger User's Guide*.

Starting Program Execution

After invoking DBG, use the RESTART subcommand to start program execution at the beginning (main entry point) of the main procedure. This subcommand also restarts the program from the beginning at any time during the debugging session. To continue from a point where the program has been stopped without going back to the beginning, use the CONTINUE subcommand.

Note

The variables initialized by a FORTRAN DATA statement or PL/I INITIAL clause are not reinitialized if RESTART is given after execution has begun.

When a program is executed under DBG, the debugger maintains control throughout execution. When the program stops or is suspended, you are returned to DBG command level and receive the DBG prompt (>), indicating that DBG is ready to receive further subcommands.

Stopping Execution

The BREAKPOINT and STEP subcommands let you interrupt program execution at a specific place. Such interruption may be very useful for examining program data.

For example, suppose that a subroutine is suspected of incorrectly handling a variable. Setting breakpoints at the entry to the subroutine and just before returning from the subroutine permits examination of the variable both before and after the subroutine has acted on it.

On the other hand, with a complicated subroutine that is branching incorrectly, it might be useful to stop at the beginning of the routine and step through it one statement at a time until suspicious behavior starts to occur.

Breakpointing

A breakpoint stops a program when it reaches a specific location. The BREAKPOINT subcommand sets new breakpoints and modifies existing ones. The format is

BREAKPOINT *breakpoint_identifier*

where *breakpoint_identifier* identifies the executable statement after which the breakpoint is to be set.

DBG allows you to use a variety of breakpoint identifiers, but the most straightforward method is to use a source line number from the current source file listing generated by the SOURCE command. For example, when debugging the program listed in the previous example, you could set a breakpoint at the line

```
300 RETURN
```

with the subcommand

```
> BREAKPOINT 25
```

If you execute a program after setting breakpoints, execution stops immediately after it reaches the first breakpoint. For example, if you now give the command

```
> RESTART
```

DBG executes your program until it reaches line 25 of the source code (statement 300 of your program) and then stops and displays the message

```
****breakpointed at $MAIN\25 ($300)
```

Note

`$MAIN` is a label that refers to the program block in which the breakpoint was set. In this case, this is the main program. In general, with programs that consist of a single file, if you use source file line numbers as breakpoint identifiers, you need not concern youself with program block labels. In other cases, you may have to specify a block label when you give subcommands like BREAKPOINT that take location identifier arguments. For more information, consult the discussions of environments, location identifiers, and breakpoints, in the *Source Level Debugger User's Guide*. `$300` refers to the FORTRAN statement label of the breakpoint.

To continue execution from this point, use the CONTINUE subcommand.

Clearing Breakpoints: The CLEAR subcommand clears a single breakpoint. For example,

```
> CLEAR 25
```

clears the breakpoint set in the previous example. CLEARALL clears all breakpoints in the program block that was executing when the program stopped.

Listing Breakpoints: The LIST subcommand displays the attributes of one breakpoint, and LISTALL displays the attributes of all breakpoints in the program block that was executing when the program stopped.

Stepping

The STEP subcommand stops program execution after a given number of statements. The format is

STEP [*value*]

where *value* is the number of statements to be executed before execution stops. If no *value* is specified, STEP executes one statement. Figure 13-2 shows the use of STEP, along with the BREAKPOINT and : (evaluate) subcommands to examine a routine in detail.

Examining and Modifying Data

The : (evaluate) Subcommand

When you breakpoint a program or step through it, you normally want to be able to examine the values of certain variables at each point. You use the : (evaluate) subcommand to examine the current value of a program variable. The use of the evaluate subcommand is illustrated in the following example:

```
> BREAKPOINT 15
> CONTINUE
   32   20

**** breakpointed at TEST\15 ($10)
> : J
J = 33
> : K
K = 21
> STEP

**** "step" completion at TEST\16 ($20)
> : J
J = 34
> : K
K = 21
> STEP

**** "step" completion at TEST\17 ($20+1)
> : J
J = 34
> : K
K = 22
```

You can also use the WATCH subcommand to keep track of variable values continuously as the program executes. The DBG example at the end of this chapter shows how to use the WATCH subcommand.

The LET Subcommand

You may also want to test new values of variables as you execute the program. Use the LET subcommand to set the value of a variable. The format is

LET ***variable = expression***

where *variable* is the name of a program variable and *expression* is any legal expression in the language of the program being debugged. The expression must evaluate to a value compatible with the type of the variable.

For example,

```
> LET J = 10
```

Note

In programs with more than one block, you may have to give a block identifier with the variable name. This usually depends on whether the variable is defined in the block that was executing when the program stopped. For more information, see the discussions of environments and variable identification in the *Source Level Debugger User's Guide*.

A Sample Debugging Session

Consider the following undebugged FORTRAN 77 program. Source line numbers are added for convenience:

```
 1: C   Total the values of all elements
 2: C   of a five-element array.
 3: C
 4:         INTEGER*2 ARRAY(5), TOTAL
 5:         DATA ARRAY/9,12,3,17,201/
 6:         TOTAL=0
 7:         DO 100 J=1,4
 8:         TOTAL=TOTAL+ARRAY(J)
 9:         I=J
10: 100     CONTINUE
11:         WRITE(1,200) TOTAL
12: 200     FORMAT('THE TOTAL OF ARRAY = ',I4)
13:         STOP
14:         END
```

Assuming that your current directory contains this program in a file called TOTAL.FTN, compile it as follows:

```
OK, F77 TOTAL -DEBUG
[F77 Rev. 22.0 Copyright (c) 1987, Prime Computer, Inc.]
0000 ERRORS [<.MAIN.> F77 Rev. 20.2]
OK,
```

The –DEBUG option informs the F77 compiler that the program to be compiled will be debugged later with DBG.

The procedure for linking the program is identical to that used without DBG, namely

```
OK, BIND
[BIND Rev 22.0 Copyright (C) 1987, Prime Computer, Inc.]
: LOAD TOTAL
: LIBRARY
BIND COMPLETE
: FILE
OK,
```

BIND saves the EPF TOTAL.RUN in your current directory. Your program has now been compiled and linked without errors. Execution succeeds, but produces an output that is clearly incorrect. The total value of the array elements is less than the value of the last element alone:

```
OK, RESUME TOTAL
THE TOTAL OF ARRAY =   41

**** STOP
OK,
```

Now you summon DBG:

```
OK, DBG TOTAL

**Dbg**   revision 22.0  (13-December-1988)

>
```

To see if the elements of the array were assigned correctly, place a breakpoint on source line 7 just before the DO loop. Enter

```
> BREAKPOINT 7
>
```

This breakpoint subcommand causes DBG to regain control immediately prior to the execution of the statement on source line 7. Begin program execution with the RESTART subcommand. The program executes until the occurrence of the breakpoint. The interaction looks like this:

```
> RESTART

**** breakpointed at $MAIN\7
>
```

The breakpoint message means that DBG has encountered the breakpoint you set on statement number 7 in the FORTRAN main program. The prompt character indicates that DBG is once again at command level awaiting a subcommand.

If you precede any variable name or expression with a colon (:) or a space, the value of the variable or the resultant value of the expression is displayed on your screen. Now check the values of the array elements.

```
> : ARRAY
ARRAY(1) = 9
ARRAY(2) = 12
ARRAY(3) = 3
ARRAY(4) = 17
ARRAY(5) = 201
>
```

You find that the values were assigned to the array correctly. Because the program is so small, you can use the WATCH subcommand to trace the changing values of the variables J and TOTAL and see where and how new values are assigned throughout execution. (When you specify two or more variables to WATCH, the variables must be separated by commas.)

Use the CLEAR subcommand to delete the breakpoint set previously on source line 7, and set a new breakpoint on source line 11 to suspend execution at the exit of the DO loop, after all values have been assigned. Now restart the program from the beginning:

```
> WATCH J, TOTAL
> CLEAR 7
> BREAKPOINT 11
> RESTART
The value of $MAIN\J has been changed at $MAIN\8
  from 0
  to   1
The value of $MAIN\TOTAL has been changed at $MAIN\9
  from 0
  to   9
The value of $MAIN\J has been changed at $MAIN\8
  from 1
  to   2
The value of $MAIN\TOTAL has been changed at $MAIN\9
  from 9
  to   21
The value of $MAIN\J has been changed at $MAIN\8
  from 2
  to   3
The value of $MAIN\TOTAL has been changed at $MAIN\9
  from 21
  to   24
The value of $MAIN\J has been changed at $MAIN\8
  from 3
  to   4
The value of $MAIN\TOTAL has been changed at $MAIN\9
  from 24
  to   41

**** breakpointed at $MAIN\11 ($100+1)
>
```

In the breakpoint message, $100+1 identifies the statement as the one following FORTRAN statement label 100.

You have solved the problem. The array index value of 5 is never assigned to J, and the value of ARRAY(5) is never added to TOTAL. Using the SOURCE subcommand, look at the DO loop index on source line 7; a 4 was assigned as the maximum iteration instead of a 5. With the QUIT subcommand, exit DBG and return to PRIMOS command level.

```
> SOURCE POINT 7
    7:          DO 100 J=1,4
> QUIT
```

After correcting, recompiling, and relinking the program, you find that it works:

```
OK, F77 TOTAL -DEBUG
[F77 Rev. 20.2 Copyright (c) 1986, Prime Computer, Inc.]
0000 ERRORS [<.MAIN.> F77 Rev. 20.2]
OK, BIND -LOAD TOTAL -LIBRARY
[BIND rev 22.0]
BIND COMPLETE
OK, RESUME TOTAL
THE TOTAL OF ARRAY =  242

**** STOP
OK,
```

Part III: PRIMOS System Facilities

14 Command Files

PRIMOS lets you replace terminal input and output with input and output from special files called command files.

- You can tell PRIMOS to save all terminal output in a command output file.
- You can save a series of PRIMOS command lines in a command input file. You can then have PRIMOS execute the commands in the file just as if you were typing them at the terminal.

Command Output Files

You can use a **command output file** (also called a **COMO** file) to save everything that appears on your terminal screen during a terminal session. A command output file contains everything that you type that is echoed to the screen and all terminal output from the system.

You use a command output file to make a record of a terminal session. After you close the file, you can display its contents at your terminal with the SLIST command. When you do this, you see your terminal session reproduced on the screen, including both your input and the system's responses.

Note

If you use ECL, only the command lines actually submitted to PRIMOS appear in COMO files by default. ECL commands, dialog, and other input are not included in COMO files. You can change this behavior with the ECL -NO_CLEAN_COMO option. See Chapter 7 for more information.

The command output file is an ordinary text file. This means you can SPOOL it to a printer or edit it with one of the Prime supported text editors.

Command output files are especially useful when you have problems during a terminal session. You can use a command output file to save input and output at the point where the problem occurred, so that you can study it or show it to someone else.

Creating a Command Output File

You create a command output file by issuing the COMOUTPUT command. The command's format is

$$\text{COMOUTPUT} \begin{Bmatrix} \textit{pathname} \; [\textit{options}] \\ \textit{options} \end{Bmatrix}$$

pathname identifies the command output file. If the file is in your current directory, just use the filename. The standard filename suffix for command output files is .COMO. The COMOUTPUT command does not supply the .COMO suffix. If you want to use the .COMO suffix, you must supply it when you give *pathname*.

If *pathname* does not exist, COMOUTPUT creates it. If *pathname* already exists, the file is overwritten, unless you specify the –CONTINUE option. The –CONTINUE option appends new terminal output to the end of the old file.

After you give the COMOUTPUT command, PRIMOS begins to send all your terminal output to the specified file. Output continues to appear on your screen, too. (If you want output to go to the command output file only, use the –NTTY option.)

For example,

```
OK, COMOUTPUT TEST.COMO
```

opens the file TEST.COMO and begins to send subsequent terminal output to it as well as to the terminal screen.

If TEST.COMO already exists when you issue the COMOUTPUT command, its previous contents are overwritten. If you want to append terminal output to the old file instead of overwriting it, use the –CONTINUE option:

```
OK, COMOUTPUT TEST.COMO -CONTINTUE
```

Closing the Command Output File

You stop recording your terminal session by issuing the COMOUTPUT command with the –END option. This option causes PRIMOS to stop sending terminal output to the command output file and closes the file. The file contains all terminal dialog beginning after the original COMOUTPUT command up to and including the COMOUTPUT –END command.

A COMO file may also be closed with the PRIMOS command

CLOSE ***pathname***

where *pathname* identifies the open command output file.

Note

The command CLOSE –ALL does *not* close a command output file; it closes all files *except* a command output file.

An open command output file is also closed when you log out or issue the INITIALIZE_COMMAND_ENVIRONMENT command.

Use the STATUS UNITS command to find out if you have an open command output file. Command output files are always assigned to a file-unit called **COMO**. For example,

```
OK, COMOUTPUT TEST.COMO
OK, STATUS UNITS


User FRED                                      S01

File    File       Open    File
Unit    Position   Mode    Type  RWlock   Treename
COMO   000000030     W      DAM   NR-1W   <DISK01>FRED>TEST.COMO

OK,
```

shows a COMO file with the pathname <DISK01>FRED>TEST.COMO.

You can have only one command output file open at a time. If you open a second command output file before closing the first one, the second COMOUTPUT command automatically closes the first file before it opens the second.

COMOUTPUT Options

Use the following options either when you first open the command output file or later during the terminal session, by issuing further COMOUTPUT commands.

Option	*Description*
–NTTY	Turns off terminal output. System output is sent to the command output file, but it does not appear on your screen. Your terminal input is still echoed unless you are using ECL.
–TTY	Turns on terminal output (default).
–PAUSE	Stops sending output to the command file, but does not close the file. You must subsequently issue a COMOUTPUT –CONTINUE or –END command.
–CONTINUE	Appends terminal output to a command output file. If the file is not open, *pathname* must precede this option. If you have suspended output to a command output file with the –PAUSE option, you don't need the *pathname*. Output is appended to the already open file.
–END	Stops sending output to the command file and closes the file. This is the recommended method of closing a COMOUTPUT file.

Note

Error messages are written to the output file and displayed at the terminal, regardless of the terminal option selected. Output sent to the terminal from other users, such as messages from the supervisor terminal, is displayed at the terminal, but not written to the output file.

Dating Command Output Files

You may find it useful to include a date and time stamp in your command output files. Use the PRIMOS DATE command. The DATE command displays the system date and time at your terminal:

```
OK, DATE
13 Dec 88 14:53:28 Thursday
OK,
```

If you issue the DATE command while a command output file is open, the date and time information is also included in the file. For example, the sequence of commands

```
COMO TEST1.COMO
DATE
  .
  .
  .
COMO -END
```

creates a file, TEST1.COMO. The first line of this file is the DATE command; the next line is the time and date of the session.

```
OK, DATE
31 Dec 87 11:38:28 Thursday
    .
    .
    .
OK, COMO -END
```

Command Input Files

A **command input file** (also called a **COMI** file) is a text file containing a series of command lines. Each line of the file represents one line of terminal input. The file can include any legal PRIMOS command lines as well as subcommands for any subsystem invoked by the command input file. Lines can also include dialog responses that you would type at the terminal during an interactive session.

You construct command input files with one of the Prime text editors. Command input files are especially useful for repetitive processes, such as compiling and linking a series of programs, building libraries, and running production jobs.

You execute command input files in three ways:

- Using the PRIMOS COMINPUT command
- As phantoms
- As Batch jobs

The COMINPUT command is introduced in this chapter. Phantom and Batch execution are described in Chapter 16.

Ending the File

The last line of a command input file depends on how you intend to execute the file.

- When a command input file is to be executed from the terminal using the COMINPUT command, the last line must be COMINPUT –END or COMINPUT –TTY. These commands return control to the terminal.
- When a command input file is to be executed as a phantom (see Chapter 16), the last line must be the PRIMOS LOGOUT command.
- When a command input file is executed as a Batch job (see Chapter 16), there is no special requirement for the last line.

Comments

You can insert comments in a command input file by using the following format:

/* *text*

where *text* is the text of your comment. When PRIMOS encounters /* in a command input file, the command interpreter ignores the rest of the line. You can use comments to add explanatory messages to your command input files. For example,

```
SLIST BENCH.MAP /* PRINT MAP FILE
```

The COMINPUT Command

Use the COMINPUT command to execute a command input file as well as to control its command flow.

The command's format is

$$\text{COMINPUT} \begin{Bmatrix} \textit{pathname} \quad [\textit{file-unit}] \\ \textit{option} \end{Bmatrix}$$

pathname specifies the command input file from which PRIMOS is to read input. If the file is in your current directory, a filename is adequate.

The standard suffix for command input filenames is **.COMI**. However the COMINPUT command does not recognize the .COMI suffix, so you must also include the suffix on the command line when you type the pathname (or filename). For example,

```
OK, COMINPUT MYFILE.COMI
```

file-unit specifies the PRIMOS file-unit number, in octal, on which *pathname* is opened. If *file-unit* is omitted, file-unit 6_8 is used by default. File-unit numbers are only necessary for nested or chained command input files. (See the section, Chaining Command Input Files, for an explanation of file-units.)

COMINPUT accepts either a pathname or an option, but not both, on a single command line.

COMINPUT Options

After PRIMOS begins to process a command input file, you can control the file's command flow with the COMINPUT options listed below. You can use only one option at a time.

The –END, –TTY, and –PAUSE options must be included in a command input file, not issued from your terminal. Conversely, you use the –START option only at your terminal. You can use the –CONTINUE option from your terminal or from within the file.

COMINPUT options are as follows:

Option	*Description*
–TTY **–END**	Either switches the command input stream to the user terminal or closes the COMI file. Use only within a command input file.
–PAUSE	Switches the command input stream to the user terminal, but does not close the command input file. Use only within a command input file.
–CONTINUE [*file-unit*]	Returns control to the COMI file after a CO –PAUSE or an error. *file-unit*, which is an octal number, must be specified if the file is opened on a file-unit other than file-unit 6 (the default file-unit). (File-units are discussed below in the section, Chaining Command Input Files.)
–START [*file-unit*]	Restarts processing of a command file after BREAK, Ctrl P, a warm start of PRIMOS, or a subsystem error. *file-unit*, which is an octal number, must be specified if the file is opened on a file-unit other than file-unit 6_8. (the default file-unit). (File-units are discussed below in the section, Chaining Command Input Files.) Use only on the command line at your terminal.

A Sample Command Input File

The following command input file, TEST.COMI, compiles and links the program TEST.FTN:

```
COMOUTPUT TEST.COMO          /* Begin test of command input file
DATE                         /* Display the time and date
FTN SQUARES                  /* Compile the program
BIND                         /* Link the program
LO SQUARES
LI
MAP FULL.MAP -FULL
MAP UN.MAP -UNDEFINED
FILE
DATE                         /* Command file test completed
COMOUTPUT -END
COMINPUT -END
```

Execute the command input file with the command

```
OK, COMINPUT TEST.COMI
```

Execution of the command input file produces the following output file:

```
OK, DATE                          /* Display the time and date
12 Dec 88 15:54:40 Monday
OK, FTN SQUARES                   /* Compile the program
0000 ERRORS [<.MAIN.>FTN-REV22.0]
0000 ERRORS [<SQUARE>FTN-REV22.0]
OK, BIND                          /* Link the program
[BIND rev 22.0]
: LO SQUARES
: LI
BIND COMPLETE
: MAP FULL.MAP -FULL
: MAP UN.MAP -UNDEFINED
: FILE
OK, DATE                          /* Command file test completed
12 Dec 88 15:54:44 Monday
OK, COMO -END
```

Errors

If the COMI file encounters an error from which it cannot recover, it returns control to the terminal, leaving the command input file open. The user may type a correct version of the line, and then resume input from the command file with the –CONTINUE option of the COMINPUT command:

COMINPUT –CONTINUE [*file-unit*]

You need to specify *file-unit* only if you are continuing a command input file that you opened on a file-unit other than the default file-unit.

Chaining Command Input Files

The –CONTINUE option of COMINPUT allows you to chain command files. That is, you can have one command input file invoke another. When the second file finishes executing, it can return control to the first, which can then continue executing. When you do this you have to take care to avoid file-unit conflicts.

File-units: Whenever PRIMOS opens a file, it assigns the file to a file-unit. You can think of a **file-unit** as a channel through which all input to and output from the file take place. File-units are identified by octal numbers. You normally don't have to be concerned with file-units, because most operations automatically assign files either to unused file-units or to default file-units that are always set aside for them.

Command input files are assigned to file-unit 6_8 by default. If you use one command input file to invoke another, you must be sure that they are assigned to different file-units. If you don't do this, both files are assigned to the default file-unit. When the second command input file is assigned to the default file-unit, it replaces the first command input file. When this happens, it is impossible to return to the first command input file when the second one finishes executing.

You can avoid this problem when chaining command input files by assigning the chained files to different file-units. You can assign command input files to any file-unit from 1_8 to 200_8.

The following example illustrates the chaining of three command files and shows how file-unit conflicts can be avoided. The command file FILE1.COMI contains the following commands:

```
COMOUTPUT CHAIN.COMO          Opens a file to record all output.
DATE                          Displays the time and date.
COMINPUT FILE2.COMI 7         Opens second COMI file on File-unit 7₈.
CLOSE 7                       Closes second COMI file.
COMOUTPUT -END                Closes the command output file.
COMINPUT -TTY                 Returns control to user terminal.
```

The command file FILE2.COMI contains these commands:

```
DATE                          Displays the time and date again.
COMINPUT FILE3.COMI 10        Opens third COMI file on File-unit 10₈.
CLOSE 10                      Closes third COMI file.
COMINPUT -CONTINUE            Returns control to the calling COMI file.
```

The command file FILE3.COMI contains the following:

```
DATE                          Displays the time and date a third time.
COMINPUT -CONTINUE 7          Returns control to calling COMI file.
```

When you invoke the first command input file with the command

```
OK, COMINPUT FILE1.COMI
```

it displays the date and then calls on FILE2.COMI. FILE2.COMI then displays the date and calls on FILE3.COMI. FILE3.COMI displays the date and returns control to FILE2.COMI. FILE2.COMI finally returns control to FILE1.COMI.

The command output file that results looks like this:

```
OK, DATE                          Displays the time and date.
09 Nov 87 10:16:28 Monday
OK, COMINPUT FILE2.COMI 7         Opens second COMI file on File-unit 7₈.
OK, DATE                          Displays the time and date again.
09 Nov 87 10:16:28 Monday
OK, COMINPUT FILE3.COMI 10        Opens third COMI file on File-unit 10₈.
```

```
OK, DATE                              Displays the time and date a third time.
09 Nov 87 10:16:28 Monday
OK, COMINPUT -CONTINUE 7                  Returns control to calling COMI file.
OK, CLOSE 10                                          Closes third COMI file.
OK, COMINPUT -CONTINUE                   Returns control to the first COMI file.
OK, CLOSE 7                                          Closes second COMI file.
OK, COMOUTPUT -END                               Close the command output file.
```

Closing Chained Command Input Files: When you chain command input files, be sure to include commands to close the called files upon returning to the calling files, using either the file-unit numbers (as in the example above) or filenames. To check for open file-units, use the STATUS UNITS command.

Do not use the command CLOSE –ALL within a command file because it closes the command file itself and displays the following error message:

```
End of file. Cominput. (Input from terminal.)
```

15

The Basics of CPL

The PRIMOS Command Procedure Language (CPL), lets you write programs that carry out sequences of PRIMOS commands and subcommands. An elementary CPL program may be a simple sequence of PRIMOS commands, much like a command input file. But CPL can also give you much more sophisticated control over the execution of PRIMOS command sequences. CPL is a high-level programming language that includes

- Control structures, such as loops and branches
- Variables
- Function calls
- Error handling and debugging facilities

This chapter provides a brief introduction to the basic features of CPL. The chapter assumes that you have some familiarity with basic programming concepts, such as variables, branching, and the like. For a complete discussion, see the *CPL User's Guide*, which contains information both for beginning and advanced programmers.

How Does CPL Work?

CPL programs are constructed with two kinds of statements: PRIMOS commands and CPL directives. The PRIMOS commands specify the operations you want PRIMOS to carry out. **CPL directives** control the flow of the program, pass arguments to the PRIMOS commands, set variable values, call subroutines, and handle errors.

CPL programs are carried out under the control of the CPL interpreter. The **CPL interpreter** is a program that reads your CPL program and passes the commands along to PRIMOS for execution. The CPL directives are instructions to the interpreter. As the interpreter reads your program, it uses the directives to decide what commands and arguments to pass to PRIMOS for execution.

When it executes a program, the CPL interpreter first evaluates variables and function calls and replaces them with their correct values. It then interprets and acts upon CPL directives. Finally, it passes the resulting commands to PRIMOS for execution.

Even a relatively simple CPL program can make many such decisions, so you can write CPL programs that carry out PRIMOS command sequences in a highly flexible manner.

Creating and Executing CPL Programs

Use one of the PRIMOS supported text editors to write CPL programs. The format is simple; you put one statement on each line. You can use indentation if you want in order to make the program easier to read.

You can add comments to a CPL program using the following format:

/* ***text***

where *text* is your comment. Whenever the CPL interpreter encounters /*, it disregards the rest of the line. You can use comments to add explanatory text that the CPL program does not try to interpret as CPL statements.

The .CPL Suffix

When you save your program, you must use a filename with the .CPL suffix:

filename.**CPL**

For example, you can call a CPL program NOTEBOOK.CPL.

Invoking CPL Programs

CPL programs are not compiled or linked. You can execute them as soon as you finish writing them. You run CPL programs interactively with either the CPL command or the RESUME command. The format is either

RESUME ***filename***

or

CPL ***filename***

where *filename* is the name of your program. You don't need to give the .CPL suffix when invoking the program as long as no similarly named files with the .RUN or .SAVE suffix exist in the same directory.

Note

If you give *filename* without a suffix when you invoke RESUME, PRIMOS searches for the following files in order:

filename.RUN
filename.SAVE
filename.CPL
filename

For example, you can invoke NOTEBOOK.CPL using

```
OK, RESUME NOTEBOOK
```

or

```
OK, CPL NOTEBOOK
```

as long as no file called NOTEBOOK.RUN or NOTEBOOK.SAVE exists in the same directory as NOTEBOOK.CPL.

You can also run CPL programs as phantoms or as Batch jobs. For details of phantom and Batch execution, see Chapter 16.

Using PRIMOS Commands in CPL Programs

The simplest CPL programs are composed entirely of PRIMOS commands. For example, you can write a CPL program that runs two programs called SUM and SORT and then lists the contents of the current directory, as follows:

```
RESUME SUM          /*This program calculates several sums
RESUME SORT         /*This program sorts the results
LD                  /*Check to see if output file exists
```

If you save this file as CALCULATE.CPL, you can then run it with the one of these commands:

```
OK, RESUME CALCULATE
```

or

```
OK, CPL CALCULATE
```

When you run this program, PRIMOS first executes a program called SUM. After SUM terminates, PRIMOS then executes a program called SORT. Finally, PRIMOS executes the LD command, listing the current directory's contents.

The interpreter ignores the comments (the text that begins with /*).

Which PRIMOS Commands Can You Use?

CPL files that consist entirely of PRIMOS commands can use the following commands:

- All compiler commands: CBL, F77, FTN, PMA, and so on.
- All commands that execute programs, such as RESUME, SEG, and BASICV.
- Any user commands that do not invoke a subsystem or initiate a dialog. For example,

  ATTACH
  LD
  CREATE
  COPY
  DELETE
  CNAME

- Commands that invoke interactive subsystems or user programs, if the you are going to supply the data or subcommands from the terminal at runtime. For example,

  ED
  BIND
  SEG
  SORT

You can also have the CPL program supply data or subcommands to interactive subsystems and programs. To do so, you use the CPL &DATA directive, explained in the section, Using CPL With Subsystems.

What PRIMOS Commands Can't You Use?

Do *not* use COMINPUT, CLOSE –ALL, or DELSEG –ALL in a CPL program. Any of these commands aborts execution of the program.

Using Variables in CPL Programs

CPL programs gain flexibility by using variables. **Variables** are character strings that can stand for any number of values. A program that uses a variable first assigns a value to the variable. Then, when the CPL interpreter processes any statement that contains the variable, the assigned value is substituted for the variable.

Variable names can have a maximum of 32 characters. They may contain only the characters A-Z, 0-9, underscore (_), and dot (.). The CPL interpreter does not distinguish between uppercase and lowercase letters.

CPL programs can assign values to variables in three ways:

- With the &ARGS directive
- With the &SET_VAR directive
- By referring to global variables

The names of variables defined with the &ARGS and &SET_VAR directives must begin with a letter. For example,

```
ARTIST_NAME
B12
```

The names of global variables must begin with a dot. For example,

```
.DEPT_CODE
```

Note

CPL variables are not strictly typed. The CPL interpreter can treat any variable value as a character string. The CPL interpreter can also treat certain character strings as integers or boolean values. For more information, see the *CPL User's Guide.*

Using the &ARGS Directive

The simplest way to assign a value to a variable is to use the CPL &ARGS directive. The &ARGS directive takes variable values from the command line when the CPL program is invoked in much the same way that abbreviations take variable values from the command line. (Abbreviations are discussed in Chapter 8.)

The format of the &ARGS directive is

&ARGS ***variable_name*** [;...***variable_name***]

where *variable_name* names a variable that is to take its value from the command line.

The following example shows how to use the &ARGS directive to get avariable's value from the command line. The example program, called F7.CPL, compiles any F77 source code and keeps a dated record in aCOMO file.

```
&ARGS FILENAME
COMO  %FILENAME%.COMO
DATE
F77   %FILENAME% -DEBUG
COMO -E
```

In this example, the &ARGS directive defines one variable, *FILENAME*. When you invoke the CPL program, you supply the value of *FILENAME* in the command line. For example,

```
OK, RESUME F7 JEFF
```

supplies the value JEFF for *FILENAME*. As the CPL program runs, the string JEFF is substituted for each occurrence of %FILENAME% in the program. So,

```
COMO %FILENAME%.COMO
```

becomes

```
COMO JEFF.COMO
```

and

```
F77 %FILENAME% -DEBUG
```

becomes

```
F77 JEFF -DEBUG
```

Note the use of the percent signs (%). When the variable *FILENAME* is defined in the &ARGS directive, the name appears without percent signs. When the variable value is used by a program statement, such as COMO %FILENAME%.COMO, the name appears between percent signs. As a general rule, when a variable is defined with CPL directives, the name appears without percent signs. When a variable value is referenced (the variable's value is used) by a program statement, the name appears between percent signs.

Note

When a variable reference is juxtaposed to another character string, with no blanks between them (as in %FILENAME%.COMO), the value of the variable is concatenated with the other string (as in JEFF.COMO). Two or more variable references may also be concatenated. For example, %FILENAME%%FILENAME% results in the string JEFFJEFF.

The value you supply in the command line that invokes the CPL program is called a **command line argument**. In the above example, the argument is the string JEFF. You use the &ARGS directive to pass command line arguments to CPL program variables.

Multiple Arguments: You can pass multiple arguments with the &ARGS directive. To do so, you separate the variable names with semicolons. For example,

```
&ARGS FILENAME; COMPILER
```

Arguments are assigned according to their position in the command line that invokes the CPL program. For example, you can write a more general CPL file, called COMPILE_ALL.CPL, that can compile FTN, F77, or PL1 source files. It reads

```
&ARGS FILENAME; COMPILER
COMO  %FILENAME%.COMO
DATE
%COMPILER% %FILENAME% -DEBUG
COMO -E
```

If you invoke this file by typing

```
OK, R COMPILE_ALL FRED FTN
```

the first argument, FRED, becomes the value of the first variable, *FILENAME*, in the &ARGS line. The second argument, FTN, is assigned to the second variable, *COMPILER*. The fourth line is then interpreted as

```
FTN FRED -DEBUG
```

Omitted Arguments

If an argument is omitted from the command line, the CPL interpreter sets its value to the null string (''). The PRIMOS command processor then removes the null string before executing the command. In the above example, the command

```
OK, R COMPILE_ALL TESTFILE
```

assigns the value TESTFILE to the variable *FILENAME*, and assigns the null string to the variable *COMPILER*. Line four then becomes

```
TESTFILE -DEBUG
```

In this case, PRIMOS can't execute such a command, so it returns you to command level with an error prompt.

Note

CPL offers several ways to deal with null arguments. These are explained in the *CPL User's Guide*.

The &SET_VAR Directive

Use the &SET_VAR directive to assign values to variables within a CPL program. The &SET_VAR directive performs essentially the same function as the assignment operator in many high-level programming languages.

The format of the &SET_VAR directive is

&SET_VAR *name* := *value*

For example,

```
&SET_VAR A := AMY
```

defines the variable *A* and gives it the value AMY.

value may also be an expression. For example,

```
&SET_VAR X := 10
&SET_VAR Y := 5
&SET_VAR Z := %x% + %y%
```

These three directives define the variables *X*, *Y*, and *Z*. *X* has a value of 10, *Y* a value of 5, and *Z* a value of 15.

Note

In CPL programs, all operators *must* be separated from their operands by one or more spaces.

Global Variables

CPL program statements can also reference global variables in an active global variable file. For example, if an active global variable file contains

```
.PROJECT                  MYDIR>PAPERWORK>NOTES
```

then the CPL program statement

```
SLIST %.PROJECT%
```

is equivalent to

```
SLIST MYDIR>PAPERWORK>NOTES
```

You can activate a global variable file either from command level, before you run the CPL program, or with a statement in the CPL program itself. However, if you run a CPL program as a phantom or in the batch environment, you must activate the global variable file with a DEFINE_GVAR statement in the CPL program itself.

For more information on global variables, see Chapter 8.

Decision Making (Branching) in CPL Programs

When a CPL program contains only PRIMOS commands (or PRIMOS commands plus variables), command lines are executed in the same sequence in which they occur in the program text. Sometimes you want to choose among alternative commands or alter the order of execution depending on conditions. Several CPL directives let you control program flow in this way.

The &IF Directive

Use the &IF directive to choose among courses of action depending on some condition. The form of the &IF directive is

&IF *test* &THEN *statement*

test is a logical test that can be answered TRUE or FALSE. *statement* is either a command or a CPL directive.

test may be constructed using any of the operators listed in Table 15-1.

TABLE 15-1
Operators

Operator	*Condition*
Arithmetic	
+	Addition, unary plus
–	Subtraction, unary minus
*	Multiplication
/	Integer division (result is truncated to integer)
Relational	
=	Equal
<	Less than
>	Greater than
<=	Less than or equal
>=	Greater than or equal
^=	Not equal
Logical	
&	And
|	Or
^	Not

test may test variables, constants, functions or expressions against each other. For example,

```
&IF %A% = 10                          Variable and constant.
&IF %A% < %B%                           Two variables.
&IF %A% < %B% + %C%                 Variable and expression.
&IF %A% + %B% = %D% + 30               Two expressions.
&IF [LENGTH %A%] < 100              Function and constant.
```

Evaluation of expressions is discussed in the *CPL User's Guide*.

How the &IF Directive Works: When the CPL interpreter reads an &IF directive, it substitutes current values for any variable references, expressions, or function calls it finds. Then it tests to see whether *test* is true or false. If *test* is true, the interpreter executes the command or directive that forms the &THEN statement.

For example, suppose you compile a program frequently, but only occasionally want to spool the listing file. You can use the &IF directive to tell the CPL program whether or not to spool the listing file. Here is a program to do it (called CNS.CPL):

```
&DEBUG &ECHO COM
    /*This program compiles and optionally spools
    /*an F77 program.
    /*Give the argument "SP" to spool the listing file.
&ARGS FILENAME; SP
    /*Open the COMOUTPUT file and compile the program
COMO %FILENAME%.COMO
DATE
F77 %FILENAME% -L %FILENAME%.LIST -XREF
    /*If desired, spool it.
&IF %SP% = SP  &THEN SPOOL %FILENAME%.LIST -AT MS3
COMO -E
```

If you give the command

```
OK, RESUME CNS JEFF SP
```

then the test, SP = SP, is TRUE, and the listing file, JEFF.LIST, is spooled. If you give the command

```
OK, RESUME CNS JEFF
```

the test is FALSE (the null string does not equal SP). In this case, the listing file is not spooled. Instead, the CPL interpreter ignores the &THEN statement, and passes on to the next line in the program (in this case, COMO –E).

The &ELSE Directive

The &IF directive may be used by itself, as in the CNS.CPL sample program above; or it may be followed by the &ELSE directive. When used by itself, &IF tells the interpreter either to execute or to ignore some statement. When the &IF and &ELSE directives are used together, they tell the interpreter to choose between two courses of action.

The format of the paired directives is

&IF *test* &THEN *statement-1*
  &ELSE *statement-2*

If *test* is true, *statement-1* is executed. If *test* is false, *statement-2* is executed. For example, suppose you compile many FTN files and a few F77 files. You can write a program (called COMPILE2.CPL) that looks like this:

```
&ARGS FILENAME; COMPILER
&IF %COMPILER% = F77 &THEN F77 %FILENAME% -DEBUG -32I
&ELSE FTN %FILENAME%
```

If you invoke the program with

```
RESUME COMPILE2 THISFILE F77
```

the test (F77 = F77) becomes true, and THISFILE is compiled by the F77 compiler in DEBUG and 32I modes. If you give any other value for the compiler argument, or if you omit the argument altogether, THISFILE is compiled by the FTN compiler in default modes.

Nested &IFs

&IF directives may be nested; either the &THEN or the &ELSE action of one &IF directive may be another &IF directive. Nested &IF statements are discussed in the *CPL User's Guide.*

&DO GROUPS

In the examples above, the &THEN and &ELSE directives execute single commands. These directives may also execute groups of commands, by using the &DO and &END directives to mark the beginning and end of each command group.

The format for &DO groups is as follows:

```
&DO
   statement 1
   statement 2
   .
   .
   .
   statement n
   &END
```

You can use an &DO group in the following way:

```
&IF test &THEN &DO
   first-group-of-statements
   &END
   &ELSE &DO
   second-group-of-statements
   &END
```

For example,

```
&ARGS MONTH
&IF %MONTH% = DEC &THEN &DO
    RESUME MONTHLY_REPORT
    RESUME END_OF_YEAR_REPORT
    RESUME XMAS_LIST
    &END
&ELSE RESUME MONTHLY_REPORT
```

Using Functions in CPL Programs

Like other high-level languages, CPL provides built-in functions to simplify frequently made tests and computations. Functions appear in CPL files in the form of function calls. In a **function call**, the function name and any arguments are enclosed in square brackets:

[***Function arg***].

When a function call appears in a command or directive, the CPL interpreter performs the required test or computation and substitutes the value produced. The function is said to **return** this value.

The NULL function

One useful CPL functions is the NULL function:

[NULL *%var%*]

where *var* is any CPL variable.

The NULL function tests for a null character string and returns the character string TRUE, if it finds one and the character string FALSE if it does not. Because the value of an omitted argument is the null string, the NULL function can be used in &IF directives to test for an omitted argument.

For example, you can use a test for a null argument to set the name of the directory listed with the LD command in the following CPL statements:

```
&ARGS WHERE
IF [NULL %WHERE%] &THEN LD MY_DIR
    &ELSE LD  %WHERE%
```

If you specify a directory name in the command line when you invoke this CPL program, the directory you specify is listed. If you don't specify a directory name in the command line, MY_DIR is listed.

The EXISTS Function

The EXISTS function determines

- Whether or not a file system object exists
- Whether it matches a specified type (file, directory, segment directory, or access category)

The format of the function call is

[EXISTS *pathname* [*type*]]

pathname is the name of a file system object.

type is one of the following:

–ANY
–FILE
–DIRECTORY
–SEGMENT_DIRECTORY
–ACCESS_CATEGORY

If *type* is present, then the EXISTS function returns the value TRUE if *pathname* does exist and is of the right type. It returns the value FALSE if *pathname* does not exist or if it is of the wrong type. If *type* is not present, the EXISTS function merely reports whether *pathname* exists. (That is, –ANY is the default when *type* is omitted.)

The following example illustrates the use of the EXISTS function. The program checks to see if the file MEMO.NEW exists. If it does exist, the program calls ED to allow the user to edit MEMO.NEW. If MEMO.NEW does not exist, the program calls ED to allow the user to edit MEMO.OLD:

```
&IF [EXISTS MEMO.NEW]  &THEN ED MEMO.NEW
    &ELSE ED MEMO.OLD
```

Using CPL With Subsystems: &DATA Groups

Many utilities, such as ED (the text editor) or BIND (a linker) require or accept subcommands. Similarly, many user programs require that data be typed in at the terminal. CPL's &DATA directive allows CPL programs to supply the data or subcommands needed by these programs and utilities.

&DATA groups resemble &DO groups in that both are groups of statements set off by an opening directive and a closing &END. In each case, the statements within the group are treated as a unit.

The form of the &DATA group is

&DATA *command*
statement-1
statement-2
.
.
.
statement-n
&END

command is the command that invokes the subsystem or utility. For example,

```
&DATA ED filename
```

invokes the Editor.

statement-1 through *statement-n* represent the commands or data to be passed to the subsystem or user program. As with all CPL statements, they may include variables, function calls, and directives.

The **&END** statement, on a line by itself, ends the &DATA group.

Here is an example of a CPL program, using an &DATA group, that compiles, links, and executes a PL/I program:

```
/*CPL program to compile, link, and execute a PL/I program
/*usage:  R CLR FILENAME
/*
&ARGS FILENAME
PL1 %FILENAME%                /*Compile program
/*
&DATA BIND                    /*Invoke BIND
  LOAD %FILENAME%             /*Provide BIND commands,
  LI PL1LIB                   /*via &data directives
  LI
  FILE
&END                          /*End of &data group
RESUME %FILENAME%.RUN         /*Execute runfile
```

Terminal Input in &DATA Groups

You may want a CPL file to invoke a subsystem or user program, give a few subcommands from within the CPL file, and then allow you to give further commands from your terminal. You do this by including CPL's &TTY directive within the &DATA group.

The format is

&DATA
 statement-1
 .
 .
 .
 statement-n
&TTY
&END

Note

You cannot use the &TTY directive if your CPL program is executed as a phantom or Batch job. In such cases, you may be able to use the &TTY_CONTINUE directive. See the *CPL User's Guide* for more information.

The &TTY directive executes after all other statements in the &DATA group have been executed. When you leave the subsystem, control returns to the CPL file.

An &TTY Example

This example shows how the &TTY directive can work with a user program. Suppose you have a program (named PURCHASE) that asks for five items of information about a customer purchase:

```
Dept. name:
Dept. number:
Customer name:
Acct. number:
Amount of purchase:
```

The hardware department can use the following CPL program (called UPDATE.CPL) to invoke the PURCHASE program and automatically supply the first two items of information:

```
&DATA R PURCHASE
   HDWR
   38
&TTY
&END
```

The example as shown could be a complete CPL program. Or, it might be part of a larger program.

A sample terminal session looks like this:

```
OK, R UPDATE
dept. name: HDWR
dept. number: 38
customer name: H.L. Smith
acct. number: 35684
amount of purchase:  536.89
OK,
```

How CPL Programs End: The &RETURN Directive

Every CPL program ends with the directive &RETURN. You may either supply this directive as the last line of the CPL file or allow the CPL interpreter to add the directive at the file's end.

You may also use the &RETURN directive to stop the program before the end of the file. For example,

```
&ARGS A
   .
   .
   .
&IF %A% > 20 &THEN &RETURN
```

```
&ELSE &DO
   .
   .
   .
   .
   .
&END
&RETURN
```

Errors in CPL Programs

CPL syntax errors halt the CPL program and return the user to command level.

PRIMOS syntax errors normally do the same. However, three exceptions to this rule exist:

- PRIMOS errors that occur during the execution of an &DATA group within the CPL program do not halt the program. Rather, they halt execution of the &DATA group and resume execution of the CPL program at the statement immediately following the &DATA group.
- Warning messages from PRIMOS or its subsystems do not normally halt execution of a CPL program.
- If you use the &SEVERITY directive (explained in the *CPL User's Guide*) in your CPL program, you can modify your program's response to PRIMOS errors and warnings.

Debugging CPL Programs

Syntax Errors

If syntax errors prevent a CPL program from executing, the interpreter displays information at your terminal or in a command output file, if one is open.

The information presented by the interpreter includes

- A line of text giving the CPL error number and the line number in the CPL program at which the error occurred.
- A full error message.
- The text of the line of source code in which the error occurred.
- A line describing the action taken by the CPL interpreter and giving the name of the program in which the error occurred. For example,

```
OK, R BLUNDER

CPL ERROR 40 ON LINE 2.

A reference to the undefined variable "FILLNAME" has been found
in this statement.
```

```
SOURCE:  como %fillname%.como

Execution of procedure terminated.  BLUNDER (cpl)
ER!
```

In this example, the program BLUNDER.CPL contained a misprint, FILLNAME, for the variable, FILENAME.

If CPL programs are halted by PRIMOS errors, then a PRIMOS error message, followed by the `ER!` prompt, is displayed.

Logic Errors

If a CPL program runs but produces incorrect results, you can use the facilities provided by CPL's &DEBUG directive to track down the errors. CPL's debugging facilities offer

- Variable watching, to display the value of a variable each time the value is set or altered.
- Echoing, to display commands and directives as they are read. (This can reveal unexpected branching.)
- A no-execute option, to allow the interpreter to walk through the CPL program without actually executing any of the commands it contains.

For full details on debugging CPL programs, see the *CPL User's Guide.*

CPL Directives Summary

CPL directives are listed below. Those marked with asterisks (*) are discussed in this chapter. All of them are discussed in detail in the *CPL User's Guide.*

Variable-handling and Argument-handling Directives

Directive	*Use*
* **&ARGS**	Defines arguments to be passed to the CPL program from the command line that executes the program.
* **&SET_VAR**	Defines a variable and sets its value, or alters the value of an existing variable.

Branching Directives

Directive	*Use*
* **&IF...&THEN...&ELSE**	Allows conditional branching.
&SELECT	Allows conditional branching among a number of specified alternatives.
* **&DO...&END**	Groups statements to be treated as a single unit syntactically. (For example, a DO GROUP may represent the action to be taken by a &THEN or &ELSE directive.)

&DO *iteration*...&END	Allows conditional iteration (that is, repeated execution) of a group of statements. CPL supports counted loops, &WHILE, &UNTIL, and &REPEAT loops. It also has two types of loops that take advantage of CPL's wildcard capabilities.
* **&DATA...&END**	Groups statements to be treated as data or subcommands for user programs or PRIMOS utilities (such as ED or BIND).
&GOTO...&LABEL	&GOTO forces an unconditional branch to the statement immediately following the &LABEL directive.
* **&RETURN**	Halts execution of program or routine and returns control to user or calling program. The CPL interpreter puts an implicit &RETURN statement at the end of each CPL program.
&STOP	Halts execution of a CPL program, whether it is used in the main program or in an internal routine.

Subroutines and User-defined Functions (Internal and External Procedures)

Directive	*Use*
&CALL	Calls (transfers control to) an internal routine.
&ROUTINE	Defines and names an internal routine.
&RESULT	Allows a CPL program to serve as a user-defined function for other CPL programs.

Execution-control Directives

Directive	*Use*
&DEBUG	Turns on (or off) CPL's debugging facility during program execution. Options to this directive specify debugging actions to be taken.
&EXPAND	Allows a CPL program to use an abbreviation file.
&SEVERITY	Defines the behavior of the CPL program (stop, continue, or call an error-handling routine) when system-defined errors or warnings occur.

Error-handling and Condition-handling Directives

Directive	*Use*
&CHECK...&ROUTINE	Checks for user-defined error conditions. Defines an internal routine to act as error-handler if the error occurs.
&ON...&ROUTINE	Defines a routine to act as a condition handler for a CPL program or routine. (For information on conditions, see Chapter 20, The Condition Mechanism.)
&REVERT	Disables a specified condition handler.
&SIGNAL	Signals a user-defined (or system-defined) condition to the condition mechanism.

16 Phantom and Batch Job Processing

PRIMOS provides phantom and Batch job processing so that you can run command files and programs without tying up your terminal. When you run a program interactively, you need to wait until it finishes execution before you can do something else at the terminal. When you use phantom or Batch execution, your terminal remains free for other work while your phantom or Batch job executes.

Phantom Execution

When you use phantom execution, PRIMOS carries out your job as if it were being run by another user. PRIMOS logs in this user, called a **phantom process**, and executes the job in the phantom's environment. Meanwhile, PRIMOS returns you immediately to command level, so that you can go on working interactively at the terminal. When the phantom job terminates, PRIMOS logs out the phantom process and informs you that the job is complete.

Note

Processes are the basic unit that PRIMOS uses to organize its work. PRIMOS establishes a process for each logged-in user, for each phantom, and for certain other system functions. Each process operates in its own environment, including such characteristics as a specific terminal line assignment, command environment limits, and the like. A major organizational task of PRIMOS is to share the system's physical resources among processes.

You can run command input files and CPL programs as phantoms. A command file or CPL program run as a phantom includes commands, program invocations, and any input data. Phantoms are especially useful for long compilations, linkings, and executions that are debugged and require no interactive terminal input. Certain PRIMOS system utilities (for example, the Batch system and SPOOL) are implemented as phantom processes.

Using Phantoms

Initiate a phantom process with the PHANTOM command. The format is

$$\text{PHANTOM } \textit{pathname} \left[\left\{ \begin{array}{c} \textit{CPL-args} \\ \textit{file-unit} \end{array} \right\} \right]$$

pathname is the name of a CPL program or COMI file. The standard filename suffix for command input files intended to be used as phantom command files is .PH. However, the PHANTOM command does not recognize the .PH suffix, so you must also include the suffix on the command line when you type the pathname (or objectname). Do not use the .PH suffix for CPL files.

If you are running a COMI file, you may include the number of the *file-unit* on which the command file is to be opened. If you omit *file-unit*, file-unit 6_8 is used. (File-units may not be specified for CPL programs, which allocate their file-units automatically.)

If a CPL program is being run as a phantom, then *CPL-args* are the arguments to be passed to the CPL program.

The System Administrator sets the maximum number of phantom users allowed on your system. The PHANTOM command checks for available phantom processes.

The following message is displayed if no process is available:

```
No phantoms are available.  filename
```

Control is then returned to PRIMOS. When a phantom process is available, the phantom user is logged in under your user ID and the following message is displayed:

```
PHANTOM is user usernumber
```

usernumber is the number assigned by PRIMOS to the phantom process.

Note

You cannot initiate a phantom process on your local system while you are attached to a directory on a remote disk. See Chapter 20 for more information on remote disks.

Phantom Operation

Once the phantom process is logged in, you are returned to PRIMOS command level. You can continue to do other work while the phantom process runs.

Input and Output: Phantom processes are like users with no terminals. Phantom processes cannot execute programs that require input from a terminal. Any instruction that requires terminal input causes the process to abort and the phantom to be logged out.

If a command file or program being run as a phantom produces output, the file or program must contain an instruction to open a COMO file. All output is then written to the COMO file. If a phantom process attempts to produce output without first opening a COMO file, the output is lost.

Initiating Other Phantoms: It is possible to initiate another phantom from a running phantom, in a manner similar to chaining COMI files. However, there is no guarantee that a phantom user process will be available when the process is requested by a command file.

Global Variables In Phantoms: A global variable file that is active for your user process is not active for a phantom process that you initiate. If you run a phantom command file or CPL program that refers to global variables, the phantom must also activate the appropriate global variable file.

Note

Phantoms you initiate do not inherit your global variables, non-default search rules, or open file-units. For example, a phantom can only refer to file-units opened by the phantom itself. Phantoms you initiate do inherit your current attach point, your origin directory, and your user ID.

Phantom Logout

The final instruction of a COMI file that is run as a phantom should be LOGOUT. If it is not, the phantom reports an abnormal termination when it is logged out. A CPL program run as a phantom does not need the LOGOUT command for a normal logout.

After completing their work, phantom users are automatically logged out. To cancel a phantom process before completion, use the following command:

LOGOUT *-usernumber*

usernumber is the user number assigned by PRIMOS.

You can log out a phantom from a user terminal only if the terminal is logged in with the same user ID as the phantom. Phantoms can also be logged out from the supervisor terminal.

Logout Notification: When a phantom logs out, notification is ordinarily sent to the terminal of the user who started the phantom. The following example shows a message generated after a normal phantom logout:

```
Phantom 87: Normal logout at 11:27
Time used: 00h 00m connect, 00m 04s CPU, 00m 05s I/O
```

Forced logout (the result of an error that halts the phantom program), logout from a terminal, or a missing LOGOUT command in a COMI file results in an abnormal logout message, as in the following example:

```
Phantom 113: Abnormal logout at 15:49
Time used: 00h 00m connect, 00m 04s CPU, 00m 03s I/O
```

In these messages, the figures following the phrase `Time used` indicate elapsed time, CPU time, and I/O time used by the phantom process.

Controlling Logout Notifications: You can use the LON command to control whether the normal phantom logout notifications are sent to your terminal. The format is

$$\text{LON} \begin{Bmatrix} \text{-OFF} \\ \text{-ON} \end{Bmatrix}$$

The command LON –OFF prevents the display of phantom logout notifications at your terminal. This can be useful if you don't want the logout notification to disturb your screen display, such as when you are using a text editor or running a COMO file.

Use LON –ON to reenable phantom logout notification at your terminal after you have given LON –OFF. If any phantoms logged out while LON –OFF was in effect, you are notified after you give the LON –ON command.

You can start a phantom and then log out before the phantom terminates. The phantom continues to run until it logs out by itself. In such cases, you don't receive logout notification. However it is possible to set up programs to record phantom logout notifications, using the subroutines LON$R and LON$CN. For information on these subroutines, see the *Subroutines Reference III: Operating System.*

Phantom Status Information

Use the STATUS ME command to get a list of all users logged in under you user ID. Phantom users are distinguished by the word `phant` in the line number field of the STATUS list. For example,

```
OK, STATUS ME
                                User No   Line No
User                              (In Decimal)      Devices (AL in
Decimal)
HAYDN                               2        0      <MUSIC3>
HAYDN                             111     phant     <MUSIC7>
<NOTES11> <SYSS44>
```

The STATUS command and its various arguments are summarized in Appendix G and fully discussed in the *PRIMOS Commands Reference Guide.*

An Example Phantom Command File

The phantom command file TEST.PH contains the following commands:

```
COMOUTPUT TESTPH.COMO     /* Open Command Output File
DATE
FTN SQUARES               /* Compile "Squares"
BIND                      /* Link "Squares"
LO SQUARES
LI
MAP FULL.MAP -FULL
MAP UN.MAP -UNDEFINED
FILE
DATE                      /* Phantom test completed
                          /* COMO -END would normally go here.
                          /* It has been omitted so the logout
                          /* sequence could be shown in the
                          /* comoutput file.
LOGOUT
```

You invoke this phantom with the command PH TEST.PH. The dialog is

```
OK, PH TEST.PH
PHANTOM is user 176
OK,
```

The phantom creates this output file, TESTPH.COMO:

```
OK, DATE
13 Dec 88 20:15:12 Thursday
OK, FTN SQUARES          /* Compile "Squares"
0000 ERRORS [<.MAIN.>FTN-REV22.0]
0000 ERRORS [<SQUARE>FTN-REV22.0]
OK, BIND                 /* Link "Squares"
[BIND rev 22.0]
: LO SQUARES
: LI
BIND COMPLETE
: MAP FULL.MAP -FULL
: MAP UN.MAP -UNDEFINED
: FILE
OK, DATE                 /* Phantom test completed
13 Dec 84 20:15:16 Thursday
OK,                      /* COMO -END would normally go here.
OK,                      /* It has been omitted so the logout
                         /* sequence could be shown in the
OK,                      /* comoutput file.
OK, LOGOUT

FRED (user 176) logged out Thursday, 13 Dec 88 20:15:16.
Time used: 00h 00m connect, 00m 05s CPU, 00m 01s I/O.
```

Batch Execution

PRIMOS provides a **Batch** subsystem for running programs detached from a user terminal. Batch provides a more regulated environment than simple phantom execution. To use the Batch subsystem, you create a file that describes the environment (minimum CPU time, input and output files, and the like) required by the program being run. This file, called a **Batch job**, can be a COMI or phantom file (described in the previous section), or a CPL file (described in Chapter 15); Batch jobs need not contain special Batch commands. After creating a Batch job, you submit it to the Batch subsystem with the JOB command.

Why Batch?

Batch is especially useful in the following situations:

- If a task runs for a long time and requires predictable input, submitting it as a Batch job frees your terminal for interactive work.

- If a task is known to monopolize system resources, submitting it as a Batch job reduces its competition with interactive users; if the job is extremely greedy, it can be configured to run only when no interactive users are present on the system.
- If a given resource (say a tape drive) is available only at certain hours of the day, users can schedule their use of that resource in advance by submitting Batch jobs to a queue that processes jobs only when that resource is available.
- If a program is to be run many times, a user can specify all the conditions under which it should be run in a Batch job; then, whenever the program must be run, the Batch job can be submitted without further ado.

How Batch Works

When a Batch job is submitted, the **Batch monitor**, the program that oversees the Batch subsystem, assigns the job to one of the **Batch queues**. A Batch queue is a list of jobs that are waiting to be run. Normally, the first job submitted to a queue runs to completion, then the second, then the third, and so on. A submitted Batch job is placed at the bottom of the list; when it reaches the top of the list, it is run.

Normally, all interactive users have a higher priority than any job in the Batch subsystem. The Batch subsystem is usually configured to run user jobs in the **background** of the system. That is, Batch jobs are run concurrently with interactive jobs, but at lower priorities. As a result, Batch jobs run more slowly when the system is heavily loaded, since they are lowest on the CPU's list of things-to-do. Within Batch, jobs running in higher-priority queues get more of the CPU's attention than jobs running in lower-priority queues. When the system is lightly loaded, Batch jobs run more swiftly.

A particular installation may have a maximum of sixteen different Batch queues; each queue may be tailored to suit a particular class of job. Your **Batch Administrator**, the person who oversees the configuration of Batch, decides how many queues your system needs and defines them with the BATGEN command. Normally, each system has some queues designed for short jobs; these jobs run at a higher priority, but are allowed a very limited allotment of CPU time in which to complete execution. Other queues, designed for ordinary jobs, have a lower priority and a higher CPU time; a few, designed for CPU-intensive jobs, run at very low priority but are allowed a lot of CPU time. A queue's priority determines how much of the CPU's attention is devoted to processing jobs in that queue.

Furthermore, Batch jobs may be held in their queues by operators, then released to run at appropriate times. Thus, extremely long jobs, such as file updates and backups, might be set up as Batch jobs during the day, then run under operator control at night; alternatively, they could be run under special Batch queues that are active only at night.

Creating a Batch Job

Batch jobs can be COMI files, CPL files, or phantoms; the Batch subsystem can deal with all of these comfortably. After you create your Batch job with an editor, try running it once with the appropriate PRIMOS command (COMINPUT, CPL, or PHANTOM) to make sure that it does not contain any syntax errors. (The Batch subsystem does not check a submitted job for syntax errors in the job itself; an incorrect job aborts silently when it reaches the erroneous line.)

Note

Batch jobs that you initiate inherit your origin directory, current attach point, and user ID. Batch jobs do not inherit your global variables, non-default search rules, or open file-units. For example, a global variable file that is active for your user environment is not active for a Batch job. If your Batch job refers to global variables, the Batch job itself must activate the appropriate global variable file.

The JOB command

To submit, monitor, or modify a Batch job, use the JOB command. The format is

JOB ***pathname*** [*options*]

The JOB command has three sets of options:

- Submit-options
- Manage-options
- Monitor-options

Use **submit-options** to control characteristics of job execution when you submit the job. Submit-options are discussed below in the section, Submitting Batch Jobs. Use **manage-options** to control the execution of a job after you submit it. The manage-options are discussed below in the section, Modifying and Canceling Batch Jobs. Use **monitor-options** to monitor the progress of your jobs in the Batch system. The monitor-options are discussed in the section, Monitoring Batch Jobs.

Submitting Batch Jobs

When you submit a job with the JOB command, the Batch subsystem accepts your job, submits it to an appropriate queue, and tells you that your job has been accepted, the home directory in which it will run, and which queue it now occupies. If you have made an error in submitting the job, or if the Batch subsystem is not processing jobs, a message is displayed. For example,

```
OK, JOB test1
[JOB Rev 22.0  Copyright (c) Prime Computer, Inc. 1988]
Your job, #00046, was submitted to queue normal-1.
Home=<RSRCH>MARY>PAYROLL
OK,
```

After you submit a job to Batch, it is assigned a **job ID**, which is a five-digit number that uniquely identifies a job. (This can be important if you submit the same job more than once.) Jobs that are submitted with no command-line options are placed in the first available queue, and assume that queue's default priority.

Submit-options: You can use the following options when you submit a job with the JOB command:

-ACCOUNTING info

Allows the user to specify a piece of text to be associated with the job. The specified text is displayed when the user types JOB -DISPLAY; it has no other effect on the job. This text may not exceed 80 characters; it cannot be an explicit register setting (octal number) or be preceded by an unquoted minus sign. If the text contains spaces, commas, or comment delimiters (/*), enclose it within apostrophes.

```
OK, JOB FRODO.CPL -ACCT 'Editorial Department'
[JOB Rev. 22.0 Copyright (c) Prime Computer, Inc. 1988]
Your job, #00001, was submitted to queue normal-1.
Home=<SHIRE>BAGEND>FRODO
OK, job -display
[JOB Rev. 22.0 Copyright (c) Prime Computer, Inc. 1988]

Job frodo.cpl(#00001), user FRODO executing (queue normal-1).
Submitted today at 3:04:44 p.m., initiated today at 3:04:46 p.m.
Funit=6, priority=5, cpu limit=None, elapsed limit=None.
Project=DEFAULT, Notify=No.
Accounting: Editorial Department
Home ufd=<SHIRE>BAGEND>FRODO
```

-ARGS *cpl-args*

Passes CPL arguments to the job being processed. -ARGS must be the last option on a command line, because JOB treats any following text (except comments) as the CPL arguments being passed.

This option forces the Batch subsystem to treat the job as a CPL file, regardless of the job's name; -ARGS can produce surprising results if the submitted Batch job is not a CPL file. The JOB command doesn't read the CPL arguments; it just passes them to the CPL file when execution of the file begins.

-COMOUTPUT *pathname*

Opens a command output file, with specified *pathname*, at the beginning of a job. This option can help you find out what went wrong if a Batch job doesn't produce the expected results. If you do not supply the -COMO option, all error messages produced by a running Batch job are silently discarded unless the Batch job itself opens a COMO file.

-CPL

Runs the submitted file as a CPL file, no matter what the filename is. (You do not need to use the -CPL option if you include the -ARGS option or if the filename ends in .CPL.)

-CPTIME $\left\{ \begin{array}{c} \textit{seconds} \\ \text{NONE} \end{array} \right\}$

Specifies the maximum amount of CPU-time (in seconds) to be allotted to the job. If the job exceeds the time limit, it is aborted. NONE requests that no time limit be placed on the job.

-DEFER *date.time*

Delays job execution to the specified date and time. You can use the date.time format: *mm/dd/yy.hh:mm:ss*. For details, see Wildcard Options in Chapter 6.

If the time to which you defer the job has passed, a warning message appears, and the job becomes eligible for immediate execution.

-ETIME $\left\{ \begin{array}{c} \textit{minutes} \\ \text{NONE} \end{array} \right\}$

Specifies the elapsed time (in minutes) to be allowed before the job is aborted. If the job exceeds the time limit, it is aborted. NONE requests that no time limit be placed on the job.

-FUNIT *number*

Specifies the file-unit to be used for command input. Permissible values range from 1 through 128.

The default value depends upon the queue to which the job is submitted. It is usually file-unit 6. (File-unit numbers must be expressed in decimal rather than octal with this option.)

You cannot use –FUNIT in CPL jobs, because CPL jobs receive dynamically assigned file-units. Attempts to use –FUNIT for CPL jobs result in the following message:

```
Illegal combination.  -FUNIT (JOB)
```

–HOME ***pathname***
Specifies the directory in which the submitted job is to run. Using this option has the same effect as providing an ATTACH command as the first line of the command file. The pathname for a –HOME option, however, may not be a null specification or a relative pathname (that is, it may not begin with *>), and may not exceed 80 characters.

–NO_COMOUTPUT
Overrides a previously-supplied –COMOUTPUT option. Include this option in an original JOB submission only if the $$ JOB command line (explained below in the section, Supplying Options Inside a Batch Job) contains the –COMOUTPUT option. Do not use this option on the same command line as the –COMOUTPUT option.

–NO_DEFER
Overrides a previously-supplied –DEFER option. Include this option in an original JOB submission only if the $$ JOB command line (explained below in the section, Supplying Options Inside a Batch Job) contains the –DEFER option. Do not use this option on the same command line as –DEFER.

–NOTIFY
Causes PRIMOS to send a message to the user when the job finishes, indicating whether the job ended normally or abnormally.

–NO_NOTIFY
Overrides a previously-supplied –NOTIFY option. Include this option in an original JOB submission only if the $$ JOB command line (explained below in the section, Supplying Options Inside a Batch Job) contains the –NOTIFY option. Do not use this option on the same command line as –NOTIFY.

–PRIORITY ***value***
Determines the job's priority in its queue. Possible values are 0 through 9, with 9 the highest (most favored) priority. The default depends on the queue.

–PROJECT ***project-ID***
Runs a job under a project ID different from the project ID under which you logged in. The job acquires the initial attach point of the specified project, so that an ORIGIN command issued in the job causes an attach to the initial attach point of the specified project ID. The job begins execution attached to the specified –HOME directory; if none is specified, the job begins execution attached to the directory from which it is submitted.

–QUEUE ***queuename***
Names the queue in which the job should be placed. (To learn the names and characteristics of queues, use the BATGEN –DISPLAY command.)

If this option is not specified, the Batch monitor places the job in the first unblocked queue that can accept the job.

–RESTART $\left\{ \begin{array}{l} \textbf{YES} \\ \textbf{NO} \end{array} \right\}$
Determines whether a job can be restarted following an ABORT or a system shutdown. The default is always –RESTART YES.

Note

If you submit a job from a directory that has a password (including your current attach point), you must use the -HOME option. Include the password in the pathname, and enclose the pathname within single quotation marks. The format is

JOB ***filename*** **-HO**ME ***'dir-name password'***

The following example illustrates a job submitted with options:

```
OK, JOB test1 -ETIME NONE -RESTART NO -ARGS TEST
```

In this example, test1 is submitted with no elapsed-time limit, as a CPL file with TEST as the argument, and is specified not to be restarted.

If, for any reason, the Batch monitor cannot accept the job as submitted, it sends the requestor information needed to resubmit the job successfully. A list of all Batch error messages is given in Appendix D.

Modifying and Canceling Batch Jobs

The following sections explain how to use JOB command **manage options** to refer to previously submitted Batch jobs, to modify them after submission, and to cancel and restart them.

Referring to Submitted Batch Jobs: The commands that modify submitted Batch jobs need an argument that specifies which job is to be altered. You can use either the jobname or the job ID. The **jobname** is the name of the file (the last element of the pathname) that you submitted to the Batch system with the JOB command. The **job ID** is the five-digit number displayed by the Batch system when it accepts your submission. (See the section Submitting Batch Jobs, above.)

If you have only one **active** (currently in a queue) job with a given name, you can supply that job's jobname when modifying the job. However, if you have more than one active copy of a job, referring to the job by its jobname is ambiguous; the Batch subsystem cannot tell which instance of the submitted job to change. In this case, you must give a job ID instead of a jobname when changing the submitted job.

For example, if file TEST has been submitted once, the command `JOB TEST -CANCEL` is adequate. But if two submissions of TEST (for example, #00057 and #00064) are active, you receive the following message:

```
Multiple jobs with this name (use internal name).
```

In this case, you must use the job ID (also known as the **internal name**) to tell the Batch system which job to cancel, as follows: `JOB #00057 -CANCEL.` You must include the pound sign (#) as part of the job ID.

Modifying Job Options: To modify a job's options after it has been submitted, use the -CHANGE option of the JOB command. If the job has already begun to execute, you need to restart the job after changing it. (See the section, Restarting Jobs, for an explanation.) The format of the JOB command with the –CHANGE option is

JOB $\left\{\begin{array}{l}\textit{jobname}\\ \textit{job-ID}\end{array}\right\}$ **–CHANGE** ***submit-option*** [***argument***]

where *submit-option* is any of the options (with an *argument*, if required) listed in the section, Submitting Batch Jobs, except –QUEUE and –PRIORITY.

The following example illustrates the use of the –CHANGE option:

```
OK, JOB test1 -DEFER 12:00
[JOB Rev 22.0  Copyright (c) Prime Computer, Inc. 1988]
Your job, #00057, was submitted to queue normal-1.
Deferred until today at 12:00:00 p.m.

OK, JOB test1 -CHANGE -NO_DEFER
[JOB Rev 22.0  Copyright (c) Prime Computer, Inc. 1988]
(Changes made) Job test1(#00057) waiting.
OK,
```

A job's –QUEUE and –PRIORITY options cannot be changed. If one or both are in error, you must cancel the job and resubmit it with the correct options.

Restarting Jobs: To restart a job already running, use the JOB command with the –RESTART option. The format is

JOB $\left\{\begin{array}{l}\textit{jobname}\\ \textit{job-ID}\end{array}\right\}$ **–RESTART**

JOB –RESTART causes a job to abort and later to restart if it is in a restartable state. If the job cannot be restarted, it aborts only. Note that JOB –RESTART works only on executing jobs. JOB –CHANGE and JOB –RESTART are often used together; you can first change an executing job's options with JOB –CHANGE, and then use JOB –RESTART to force the job to abort and begin executing with the new options.

The following example illustrates changing and restarting an executing job:

```
OK, JOB SCORES -CHANGE -HOME resrch>stats>newstats
OK, JOB SCORES -RESTART
```

The JOB –CHANGE option modifies the job's options; the JOB –RESTART command terminates execution and then flags the job as ready for restarting under its new conditions.

Note

Note the difference between the –RESTART option with the YES and NO arguments (a submit-option, described in the previous section) and the –RESTART option with no arguments (a manage-option, described in this section). The YES and NO arguments indicate whether or not a job may be restarted. The –RESTART option without arguments aborts the job and attempts to restart it.

Canceling Jobs: To prevent the running of a Batch job that is waiting in the queue, use the –CANCEL option. The format is

$$\textbf{JOB} \left\{ \begin{array}{l} \textit{jobname} \\ \textit{job-ID} \end{array} \right\} \textbf{-CANCEL}$$

This command does not halt a job that has started to run. To halt a running job, use the –ABORT option (explained in the next section). In the following example, a job is submitted and then canceled:

```
OK, JOB test1 -DEFER 12:00
[JOB Rev 22.0  Copyright (c) Prime Computer, Inc. 1988]
Your job, #00060, was submitted to queue normal-1.
Deferred until today at 12:00:00 p.m.
Home=<SALES>RICHTER>BATCHUPDA
OK, JOB test1 -CANCEL
[JOB Rev 22.0  Copyright (c) Prime Computer, Inc. 1988]
Job TEST1(#00060) cancelled.
OK,
```

Aborting Jobs: To terminate execution of a job already running, use the –ABORT option. The format is

$$\textbf{JOB} \left\{ \begin{array}{l} \textit{jobname} \\ \textit{job-ID} \end{array} \right\} \textbf{-ABORT}$$

This command cancels a waiting, deferred, or held job and forces a running job to log itself out immediately. For example,

```
OK, JOB test1
[JOB Rev 22.0  Copyright (c) Prime Computer, Inc. 1988]
Your job, #00038, was submitted to queue normal-1.
Home=<SALES>RICHTER>BATCHUPDA
OK, JOB test1 -ABORT
[JOB Rev 22.0  Copyright (c) Prime Computer, Inc. 1988]
Job TEST1 (#00038) aborted.
```

Notes

Remember that JOB –ABORT and JOB –RESTART accept a *jobname* in place of a *job-ID* only if the user has only one active job of that name. See the section Referring to Submitted Batch Jobs, above, for details.

Only one of the manage-options (-CHANGE, –CANCEL, –ABORT, and –RESTART) may appear on a JOB command line at one time. For example, JOB TEST –ABORT –RESTART is illegal.

Supplying Options Inside a Batch Job

Use the $$ JOB command line to include command-line options for a Batch job inside the job's command file. Any or all of the submit-options, listed in Table 17-1, can be used on the

$$ JOB command line. You may not use any of the manage-options in the $$JOB command line. The format of the $$ JOB command line is

```
$$ JOB { *       } [submit-options]
       { user-ID }
```

If a specific *user-ID* is given on the $$ JOB command line, only a user logged in with that user ID can submit the file. An asterisk (*) indicates that any user can submit the file.

Assume that user Richter adds the following line at the beginning of the Batch job test1:

```
$$ JOB RICHTER -COMOUTPUT test1.como -queue express-1 -CPTIME none
```

From now on, each time Richter types `JOB test1`, the job produces the COMO file, test1.COMO, runs in the express-1 queue, and has no CPU-time limit.

If you expect to use the same JOB options whenever a job is submitted, include those options in a $$ JOB command line; if you expect to use different options with the same Batch job, specify the options independently whenever you submit the job with the JOB command.

However, if you are submitting a job and would like its options to be different from those included in a $$ JOB command line, you can override the $$ JOB options with your JOB command. For instance, if user RICHTER decides to submit test1 to a queue that requires a CPU-time limit, she can submit test1, as follows:

```
JOB test1 -CPTIME 180
```

Options given on the JOB command line at PRIMOS level override options included in a $$ JOB command line within your file.

Note

A command file that includes a $$ JOB command line does not have to be run as a Batch job. If you run such a file interactively, PRIMOS ignores the $$ JOB line.

The combination of the JOB command and the $$ JOB command line may contain a maximum of 160 characters. If your JOB command exceeds the number of characters visible on the terminal screen, continue input *without* pressing the return key.

Monitoring Batch Jobs

You may request information on the progress of your own jobs within the Batch system in three ways:

- Use the –NOTIFY option when submitting a job
- Use the JOB command with the –STATUS or –DISPLAY options at any time after you submit the job
- Use the MESSAGE command to have your Batch job send you messages at specific points during its execution

Using the JOB –NOTIFY Option: A job submitted with the –NOTIFY option informs you at job termination whether your job has completed or aborted. For example,

```
OK, JOB test1 -NOTIFY
[JOB Rev 22.0  Copyright (c) Prime Computer, Inc. 1988]
Your job, #00131, was submitted to queue normal-1.
OK,
Job test1 for RICHTER (#00131) completed.
```

If you forget to submit a job with the –NOTIFY option, you can add the option with the JOB –CHANGE command; similarly, you can use

JOB *jobname* –CHANGE –NO_NOTIFY

to turn off a previously-specified –NOTIFY option.

Using JOB –STATUS and JOB –DISPLAY: After you submit a Batch job, you can find out what is happening to the job with one of the monitor-options, –STATUS and –DISPLAY. The format is

$$\textbf{JOB}\left[\begin{Bmatrix}\textit{jobname}\\ \textit{job-ID}\end{Bmatrix}\right]\begin{matrix}\textbf{–STATUS}\\ \textbf{–DISPLAY}\end{matrix}$$

The *jobname* and *job-ID* arguments allow you to specify the jobs on which you want information, as follows:

Argument	*Description*
jobname	Requests information on all active jobs with the name *jobname* (useful with multiple submissions of a file).
job-ID	Requests information on one unique instance of a job.

If no argument is supplied, the Batch subsystem supplies information on *all* the user's active jobs.

Option	*Description*
–STATUS	Displays the job's *jobname* and *job-ID*, the name of the queue in which it is placed, and its execution status: held (delayed by operator request), waiting, deferred, or executing.
–DISPLAY	Displays the same information as –STATUS as well as values for all JOB and $$ JOB command options, both those specified by the user and those assigned from queue-specified defaults. –DISPLAY also shows the home directory of the job, including any directory passwords in the pathname.

Example of the JOB –STATUS Command: Assume that user RICHTER submitted a job named test1. The job is stored in the subdirectory <SALES>RICHTER>BATCHUPDA. The JOB –STATUS command generates the following report:

```
OK, JOB test1
[JOB rev 22.0  Copyright (c) Prime Computer, Inc. 1988]
Your job, #00044, was submitted to queue normal-1.
Home=<SALES>RICHTER>BATCHUPDA
OK, JOB -STATUS
[JOB Rev 22.0  Copyright (c) Prime Computer, Inc. 1988]
No waiting, deferred, or held jobs; 1 executing job.
OK,
```

Example of the JOB –DISPLAY Command: Assume, as above, that user RICHTER submitted a job named test1. The JOB –DISPLAY command generates the following report:

```
OK, JOB -DISPLAY
[JOB Rev 22.0  Copyright (c) Prime Computer, Inc. 1988]
Job test1(#00044), user RICHTER executing (queue normal-1).
Submitted today at 1:40:08 p.m., initiated today at 1:40:09 p.m.
Funit=6, priority=5, cpu limit=None, elapsed limit=None.
Project=DEFAULT, Notify=No.
Home directory=<SALES>RICHTER>BATCHUPDA
OK,
```

If a job is restarted, the following message appears after the `Submitted...` line in the report above:

```
(This job has already executed n times).
```

n is the number of previous executions (or the number of restarts).

Using the MESSAGE Command: Another way to monitor your Batch jobs is to have the jobs send messages to you announcing the completion of key portions of the job. To do this, include the MESSAGE command in your Batch job. (To be notified when the job as a whole has completed, submit jobs with the –NOTIFY option, described in the section, Submitting Batch Jobs.)

Messages From CPL Programs: CPL programs put their messages in &DATA groups. The format is

&DATA MESSAGE *user-ID*
  text of message
&END

For example,

```
&DATA MESSAGE BEECH
   Customer list update completed
&END
```

If the message recipient is not logged in, the CPL job aborts. To avoid this, use the &SEVERITY &ERROR &IGNORE directive.

Messages From COMINPUT Files: Include message text in COMINPUT files as comment lines:

MESSAGE ***user-ID***
/* text of message

For example,

```
MESSAGE BEECH
/* Customer update completed
```

The comment characters preceding the message prevent errors from occurring if the recipient of the message is not logged in when the message is sent.

Monitoring the Entire Batch Subsystem

You may request information on usage of the entire Batch subsystem with the command:

$$\textbf{BATCH} \begin{Bmatrix} \textbf{-STATUS} \\ \textbf{-DISPLAY} \end{Bmatrix}$$

The BATCH –STATUS command displays a one-line summary giving the number of waiting, deferred, or held jobs, the number of queues that have waiting, deferred, or held jobs, and the number of executing jobs. If there are waiting, deferred, or held jobs as well as executing jobs, the total number of active Batch jobs is also displayed. For example,

```
OK, BATCH -STATUS
[BATCH Rev 22.0  Copyright (c) Prime Computer, Inc. 1988]
3 waiting, deferred, or held jobs; 1 executing job.
```

The BATCH –DISPLAY command displays information on Batch usage in two tables. The first table displays the number of jobs waiting or held in each queue. The second table lists the number of jobs currently executing and identifies each by user ID, job ID number, phantom user number, and queue. An example follows.

```
OK, JOB batchupda
[JOB Rev 22.0  Copyright (c) Prime Computer, Inc. 1988]
Your job, #00031, was submitted to queue normal-1.
Home=<SALES>RICHTER>BATCHUPDA
OK, BATCH -DISPLAY
[BATCH Rev 22.0  Copyright (c) Prime Computer, Inc. 1988]
No queues have waiting, deferred, or held jobs.

1 currently running job:

  User    Jobid#   # Queue
-------- ------ --- --------
RICHTER  #00031 126 normal-1
```

Monitoring Characteristics and Availability of Queues: You can get information about your site's Batch queues with the BATGEN command:

```
BATGEN { -STATUS                  }
       { -DISPLAY [queuename]     }
```

BATGEN –STATUS: The BATGEN –STATUS command lists the currently defined queues, in the order established by your System Administrator.

```
OK, BATGEN -STATUS
[BATGEN Rev 22.0  Copyright (c) Prime Computer, Inc. 1988]

Queue:       Status:
------------ -------------------
express-1    unblocked uncapped
express-2    unblocked uncapped
normal-1     unblocked uncapped
normal-2     unblocked uncapped
background-1 unblocked uncapped
background-2 unblocked uncapped (inactive)
```

A queue's **status** is described by one or more of the following words:

Status	*Meaning*
blocked	Not accepting new jobs
capped	Not executing jobs
unblocked	Accepting new jobs
uncapped	Executing jobs
inactive	Not executing jobs; will execute jobs when a specified **active window** is reached

A queue may be both blocked and capped, in which case it neither accepts new jobs nor processes any jobs already in the queue; it may also be blocked only or capped only. By default, all queues are unblocked, uncapped, and active.

BATGEN –DISPLAY: The BATGEN –DISPLAY command identifies and gives full characteristics of the *queuename* specified. If you omit the *queuename*, BATGEN –DISPLAY gives the characteristics of all queues. The following example shows the display of all queues:

```
OK, BATGEN -DISPLAY

Queue name = express-1, unblocked, uncapped.
Active window = FULL
Default cptime=121, etime=6, priority=9;
Maximum cptime=120, etime=5; Funit=6;
Delta rlevel=0; Timeslice=10;
```

```
Queue name = express-2, unblocked, uncapped.
Active window = FULL;
Default cptime=301, etime=16, priority=9;
Maximum cptime=300, etime=15; Funit=6;
Delta rlevel=0; Timeslice=10;

Queue name = normal-1, unblocked, uncapped.
Active window = FULL;
Default cptime=None, etime=None, priority=5;
Maximum cptime=None, etime=None; Funit=6;
Delta rlevel=1; Timeslice=20;

Queue name = normal-2, unblocked, uncapped.
Active window = FULL;
Default cptime=None, etime=None, priority=5;
Maximum cptime=None, etime=None; Funit=6;
Delta rlevel=1; Timeslice=20;

Queue name = background-1, unblocked, uncapped.
Active window = FULL;
Default cptime=None, etime=None, priority=5;
Maximum cptime=None, etime=None; Funit=6;
Delta rlevel=2; Timeslice=40;

Queue name = background-2, unblocked, uncapped.
Active window = 20:00 - 07:00;
Default cptime=None, etime=None, priority=5;
Maximum cptime=None, etime=None; Funit=6;
Delta rlevel=IDLE; Timeslice=50;
OK,
```

The display entries give the following information:

Label	*Meaning*
Queue name	Identifies the queue and tells whether jobs are being accepted and executed.
Active window	Specifies a time span when queue is active. Your Batch Administrator uses the BATGEN ACTIVE_WINDOW subcommand to set a time window for the queue's activity. Jobs may be submitted to such a queue at any time, but are executed only during the queue's active window. If BATGEN –DISPLAY lists a queue's active window as FULL, then that queue is active at all times.
Default cptime and etime	Specifies the default CPU-time limit (**cptime**) in seconds and elapsed-time limit (**etime**) in minutes for jobs that don't specify their own CPU-time and/or elapsed-time options.
Maximum cptime and etime	Specifies the longest time allowed for any job running in the queue.
Priority and Funit	Specifies default values for priority and file-unit.

Delta rlevel and Timeslice

Refers to runtime priorities. Queues with high delta rlevels and large timeslices are best for long jobs; queues with low delta rlevels and short timeslices are best for short jobs. The larger the delta rlevel number, the lower the priority.

IDLE is the lowest priority that can be specified for delta rlevel; a queue with delta rlevel IDLE executes jobs only when no higher-priority PRIMOS processes are waiting for execution.

17 File-handling Utilities

PRIMOS supports several file-handling utilities. These utilities allow you to

- Sort one or more unsorted files into one sorted file
- Merge several sorted files into one sorted file
- Compare files with each other
- Resolve differences between files
- Join files sequentially

This chapter describes basic operations on ASCII (text) files, although the SORT utility can also process binary files.

ASCII File Structure

The basic unit of file structure is the record. In an ASCII file, a **record** is a line of text (a string of ASCII characters terminated by a linefeed character). The file-handling utilities operate on files one record at a time; they compare and sort files line by line. SORT can also operate on data items within a line. The discussion of SORT explains how.

Sorting Files

The SORT command sorts as many as 20 files, on as many as 50 keys, into a single output file. SORT preserves the order of input for records with equal keys; that is, it is a **stable sort**.

Notes

Most sorts are done on ASCII files (also called **compressed files**), such as those created by the text editor (ED). The following discussion emphasizes how to do ASCII sorts. In addition, SORT can process uncompressed files, variable-length files (also called **binary files**), and fixed-length files. The basic format for using SORT is the same for every file type, but details vary from type to type.

SORT can also use the EBCDIC collating sequence to sort files. The *PRIMOS Commands Reference Guide* contains complete information and sorting instructions for each file type and collating sequence.

Using SORT

To use SORT, you provide information in a three-step or four-step sequence, as follows:

1. Give the SORT command (and any desired options).
2. Specify the sort files and number of sort fields, either by a simple parameter list or by the use of keywords.
3. Specify the starting and ending columns of sort fields (keys).
4. If you specified –MERGE in Step 1, enter additional filenames.

SORT normally requests the information it wants at steps 2, 3, and 4. However, once you are familiar with the dialog, you can suppress it by using the –BRIEF option on the command line. If you specify –BRIEF, give the information line by line in the same order that SORT asks for it. Refer to the sample sort given below for an example of the SORT dialog.

The SORT Command

To invoke SORT, give the SORT command, either by itself or accompanied by one to four options:

```
       ┌ -BRIEF          ┐
       │ -SPACE          │
SORT   │ -MERGE          │
       │ ┌ -TAG     ┐    │
       └ └ -NONTAG  ┘    ┘
```

SORT options are as follows:

Option	*Meaning*
–BRIEF	Inhibits display of SORT messages and prompts
–SPACE	Deletes any blank lines in the input file(s)
–MERGE	Specifies a merge of presorted files
–TAG	Specifies a tag sort (the default, described below)
–NONTAG	Specifies a nontag sort (described below)

Specify **–TAG** when you have large files to be sorted. For unordered files, tag sorts are faster than nontag sorts. Internally, the tag sort stores input records separately from the key data. After all keys have been sorted and merged, the corresponding records are then located and output.

Specify **–NONTAG** for smaller or well-ordered input files. Internally, the nontag sort stores each input record with its sort key in the work file. This eliminates the search for each record after merging, but requires more disk space.

When you give the SORT command without the –BRIEF option, SORT requests the following information. (If you use the –BRIEF option, give the information in the same order.)

- The name of the file to be sorted
- The name of the output file to be created
- The number of keys for the sort (default is 1)

Simple File and Key Specifications

The simplest type of sort reads one unsorted ASCII file and creates another sorted ASCII file. SORT can operate on sort keys within a line. Since the file itself contains no information about where one item begins and another ends within a line, you must provide this information by defining **key fields** that correspond to the locations of the data items. You do this by giving SORT the starting and ending column numbers of each field. Remember that SORT has no other way of recognizing valid data items within a line, so you need to be sure that the correct data falls in each field in your file.

To specify a simple sort, list the filenames and number of keys (if greater than 1) on one line, then list the starting and ending columns for each key field on a separate line. The following example sorts a list of names and addresses in ascending order, with the entire entry of 80 characters as the sort field:

```
OK, SORT -BRIEF
JUMBLED.NAMES   NEAT.NAMES
1   80
```

Unless the –MERGE option has been specified, sorting begins when you enter the last pair of column numbers. When the sort is complete, SORT displays the number of passes needed for the sort and the number of items (that is, lines) placed in the output file, and then returns to PRIMOS command level.

Other File Specifications

If you are sorting more than one file, give all filenames plus the number of key fields on a single line in the following format:

–INPUT *inputfile* [...–INPUT *inputfile*] –OUTPUT *outputfile* –KEYS *n*

For example,

```
OK, SORT -BRIEF
-INPUT CHAOS.1 -INPUT CHAOS.2 -OUTPUT ORDER -KEYS 2
1 10
15 20 R

BEGINNING SORT

PASSES        2           ITEMS       10

[SORT-REV22.0]

OK,
```

Key Code Specifications

If you want to sort a key field by some code other than simple ASCII, specify the type of sort by including a key code after the ending column number (separated by one space). You can specify a reverse sort (in descending order) by typing R after the ending column number (separated by one space).

Other key codes for ASCII files (compressed and uncompressed) include **A** and **AU** for alphanumeric data, **U**, **LS**, **TS**, **LE** and **TE** for numeric data. This section describes the most commonly used key codes. Other keys are discussed in the *PRIMOS Commands Reference Guide*.

Alphanumeric keys: The two alphanumeric key codes are ASCII (A), which sorts in a strict ASCII sequence, and ASCII, uppercase and lowercase (AU), which sorts all alphanumeric characters as if they were uppercase. (The ASCII sequence is given in Appendix C.) The default key type is strict ASCII (A).

Given the four words, APPLE, alphabet, WHY, and whynot, ASCII (A) produces

```
APPLE
WHY
alphabet
whynot
```

AU produces

```
alphabet
APPLE
WHY
whynot
```

Numeric keys: Three common numeric key codes for ASCII sorts are

Key	*Meaning*
U	Unsigned: numbers without plus or minus signs.
LS	Leading Sign: numbers preceded by plus or minus signs. (Numbers without signs are considered positive.)
TS	Trailing Sign: numbers followed by plus or minus signs. (Numbers without signs are considered positive.)

The Leading Embedded (LE) and Trailing Embedded (TE) keys, which have the sign embedded in the numeral, are explained in the *PRIMOS Commands Reference Guide*.

Here is an example of a sort on an LS key:

```
OK, SORT -BRIEF
NUMBERS NUMBERS.1
1 10 LS
```

```
BEGINNING SORT

PASSES        2           ITEMS        7

[SORT-REV22.0]

OK, SLIST NUMBERS.1
-9999
-8205
-6783
 4114
+5483
 8265
+9765

OK,
```

Additional Filenames for the –MERGE Option

After key fields have been specified with the –MERGE option, SORT asks for the number of additional files to be merged. If you have already listed all input files with the –INPUT format, this number is 0. Otherwise, give the number of additional files and then the names of the files, one name per line. When the last name is entered, the mergesort begins. When the mergesort is complete, SORT displays the number of passes and returns to PRIMOS command level.

A Mergesort Example

Suppose that you have two transaction files in which each line (record) has the following format: a transaction number in columns 1 through 5, a credit or debit notation in column 6, a customer name in columns 8 through 17, a customer ID number in columns 19 through 25, and other data in the remaining columns. Each file has been sorted by customer name, customer ID, and transaction number (in reverse order, so that most recent transactions come first). The full SORT dialog to merge the two files, sorting on the same three keys, is as follows:

```
OK, SORT -MERGE
SORT PROGRAM PARAMETERS ARE:
  INPUT TREE NAME -- OUTPUT TREE NAME FOLLOWED BY
  NUMBER OF PAIRS OF STARTING AND ENDING COLUMNS.
CUST.CREDITS CUST.ACCTS 3
  INPUT PAIRS OF STARTING AND ENDING COLUMNS
  ONE PAIR PER LINE--SEPARATED BY A SPACE.
  FOR REVERSE SORTING ENTER "R" AFTER DESIRED
  ENDING COLUMN--SEPARATED BY A SPACE.
  FOR A SPECIFIC DATA TYPE ENTER THE PROPER CODE
  AT THE END OF THE LINE--SEPARATED BY A SPACE.
     "A"  - ASCII
     "I"  - SINGLE PRECISION INTEGER
```

```
      "F"  - SINGLE PRECISION REAL
      "D"  - DOUBLE PRECISION REAL
      "J"  - DOUBLE PRECISION INTEGER
      "U"  - NUMERIC ASCII,UNSIGNED
      "LS" - NUMERIC ASCII,LEADING SEPARATE SIGN
      "TS" - NUMERIC ASCII,TRAILING SEPARATE SIGN
      "LE" - NUMERIC ASCII,LEADING EMBEDDED SIGN
      "TE" - NUMERIC ASCII,TRAILING EMBEDDED SIGN
      "PD" - PACKED DECIMAL
      "AU" - ASCII, UPPER & LOWER CASE SORT EQUAL
      "UI" - UNSIGNED INTEGER
   DEFAULT IS ASCII.
8 17
19 25
1 5 R
INPUT THE NUMBER OF ADDITIONAL FILES TO BE MERGED. (MAX=    10): 1
  INPUT FILES TO BE MERGED, ONLY ONE PER LINE.
CUST.DEBITS

BEGINNING MERGE

PASSES       1          ITEMS       10

[SORT-REV22.0]

OK, SLIST CUST.ACCTS
89424+ Jones      BR9438   other data about transaction
81884- Jones      BR9438   other data about transaction
12345+ Jones      BR9438   other data about transaction
67340- Jones      XL1489   other data about transaction
54936+ Jones      XL1489   other data about transaction
49480- Jones      XL1489   other data about transaction
86889+ Smith      CS4192   other data about transaction
29622+ Smith      CS4192   other data about transaction
23220- Smith      CS4192   other data about transaction
21220+ Smith      CS4192   other data about transaction

OK,
```

Comparing Files

The PRIMOS command CMPF permits the simultaneous comparison of as many as five ASCII files of varying lengths. The format is

CMPF *file-1 file-2* [*file 3...file-5*] [*options*]

The CMPF command treats the first file, *file-1*, as the original file and compares the other files to it. It then produces output indicating which lines have been added, changed, or deleted from the original to produce the other files.

The following options may be specified:

Option	*Function*
–BRIEF	Suppresses display of differing lines of text of files being compared. Only identification letters and line numbers are displayed.
–MINL *number*	Sets the minimum *number* of lines that must match after a discrepancy between files is found. Needed to resynchronize file comparison. Default is 3 lines.
–REPORT *filename*	Produces a file with specified *filename*, containing the differences found between compared files (in lieu of displaying them at the terminal during the comparison process).
–STOP	Stops the comparison as soon as a difference between files is found.

After CMPF discovers a difference between the original file and another specified file, CMPF attempts to resynchronize the files for comparison. This occurs only when a certain number of lines match in all the files being compared. The default value is 3, but can be changed with the –MINL option. The comparison process continues until another difference is found.

When line differences are reported, either at the terminal or in a report file, each line from the original file is indicated by the letter A, followed by the number of the line containing discrepancies. The corresponding lines of other files are indicated in the same manner, using letters B through E, respectively.

For example, consider the following two files:

FILEA	*FILEB*
The	The
quick	swift
brown	red
fox	fox
jumps	jumps
over	over
the	the
lazy	dog
dog	

A CMPF comparison of these two files works as follows:

```
OK, CMPF FILEA FILEB
[CMPF 22.0]

A2          quick
A3          brown
CHANGED TO
B2          swift
B3          red
A8          lazy
DELETED BEFORE
B8          dog
```

```
COMPARISON FINISHED.
2 DISCREPANCIES FOUND.

OK,
```

Merging Text Files

The MRGF command merges as many as five ASCII files. The format is

MRGF ***file-a file-b*** [***file-c ... file-e***] **–OUTF** ***outputfile*** [***options***]

The first file specified is treated as the original file, and it is assumed that changes have been made to this file to produce the other files. Pathnames may be used to specify files to be merged. Unchanged lines of text and nonconflicting changes between files are automatically copied to the output file, *outputfile*. The original source files (*file-a*, *file-b*, and so on) are not changed; *outputfile* is built from them.

When corresponding lines of text in the files differ, the MRGF program asks you to resolve the conflicts. Conflicts are resolved by entering an interactive mode in which you can specify the contents of the output file. In this mode, the subcommand *x* causes all the queried lines from *file-x* to be inserted into *outputfile*, where *x* = A, B, C, D, or E; A signifies *file-a*, B signifies *file-b*, and so on. The subcommand *xn* causes line *n* from *file-x* to be inserted.

You can insert new text by entering a blank line at the terminal (thus placing MRGF in input mode), typing the new text, and then typing another blank line. No text editing can be performed on lines thus input, and no expansion of tab characters is done. The lines must be entered character-for-character as they are to appear.

The subcommand GO terminates editing and proceeds with the merge.

The options taken by the MRGF command are similar to those for the CMPF command. An additional option, –FORCE, causes *file-b* to be the preferred file if conflicts exist between several files. No MRGF interactive dialog is generated when conflicts arise if the –FORCE option is specified. *file-b* is assumed to be correct, and the other files are forced to comply with it.

For example, consider the following two files:

MAXIM	*NOMAXIM*
`A`	`The`
`rolling`	`rolling`
`stone`	`old`
`gathers`	`oaken`
`no`	`bucket`
`moss`	`holds`
	`no`
	`milk`

A merging of these two files proceeds as follows:

```
OK, MRGF MAXIM NOMAXIM -OUTF MIX -MINL 1
[MRGF 22.0]

A1         A
CHANGED TO
B1         The
EDIT.
B
GO

A3         stone
A4         gathers
CHANGED TO
B3         old
B4         oaken
B5         bucket
B6         holds
EDIT.
B3
A3
B6
Return
INPUT.
it would seem
Return
EDIT.
GO

A6         moss
CHANGED TO
B8         milk
EDIT.
A
GO

MERGE FINISHED.
3 MANUAL CHANGES.
NO AUTOMATIC CHANGES.

OK,
```

The subcommand B selects NOMAXIM's version of the differing line in the first edit group and inserts it into MIX. The subcommand GO returns MRGF to the merge activity.

The subcommands B3, A3, and B6 insert these lines (`old`, `stone`, and `holds`), in this order, into MIX. An extra Return puts MRGF in input mode to accept the string `it would seem`. A second extra Return puts MRGF back in edit mode. The subcommand GO again returns MRGF to the merge activity.

The subcommand A selects MAXIM's version of the differing line in the next edit group and inserts it into MIX. The subcommand GO once again returns MRGF to the merge activity.

The output file MIX now contains the following:

```
MIX

The
rolling
old
stone
holds
it would seem
no
moss
```

More detailed information on MRGF appears in the *PRIMOS Commands Reference Guide*.

Joining Several Files Sequentially

The CONCAT command concatenates files into a single file. A common use of CONCAT is to concatenate several files into one file, which can then be printed with the SPOOL command.

The format for CONCAT is

CONCAT ***new-filename*** **[*options*]**

The *options* govern the format of the printout and the disposition of the files. For details, see the discussion of CONCAT in the *PRIMOS Commands Reference Guide*.

When you give the CONCAT command without options, CONCAT goes into input mode. It asks for the names of the files to be concatenated, and displays a colon prompt. Type the filenames, one per line. A null line Return signals the end of list. CONCAT then goes into command mode, and displays a right-angle prompt. Type `QUIT` to end the session. (You can also type `INPUT` to return to input mode, or you can give various formatting commands, which are explained in the *PRIMOS Commands Reference Guide*.)

For example,

```
OK, CONCAT TRIPLET
[CONCAT Rev 22.0]

Enter filenames, one per line:
: FIRST
: SECOND
: THIRD
: Return

> Q

OK,
```

If the file TRIPLET already exists, CONCAT prompts

```
OK TO MODIFY OLD TRIPLET?
```

Answering NO returns you to PRIMOS command level. Answering YES prompts a second question:

```
OVERWRITE OR APPEND:
```

Answering OVERWRITE causes CONCAT to replace the old TRIPLET with a new version. Answering APPEND preserves the existing contents of TRIPLET and adds the new files at the end.

18 Tapes

You can back up disk files to magnetic tape and restore tape files to disk using the MAGRST and MAGSAV utilities. This chapter describes the basic steps required to access a tape drive and carry out backup and restore operations. Further information appears in the *Data Backup and Recovery Guide*, the *MAGNET User's Guide*, and the *PRIMOS Commands Reference Guide*.

To transfer files between tape and disk, you need to carry out the following steps:

1. Obtain exclusive use of a tape drive with the ASSIGN command.
2. Transfer the files with the appropriate utility.
3. Relinquish exclusive use of the tape drive with the UNASSIGN command.

Assigning Tape Drives

Your System Administrator decides how magnetic tape drives may be assigned. The three possibilities are

- Each user can assign a tape drive from any terminal; operator intervention is necessary only for processing special requests. This is the default mode.
- Each user must send all assignment requests through the operator, who controls all access to tape drives. The operator then sends messages to the user, indicating the status of the assignment request.
- Tape drive assignment from any user terminal is strictly forbidden. This feature restricts access to tape drives in security-conscious environments; it is also used when the operator is not available to process requests.

ASSIGN Command Formats

If your system allows you to make or request tape drive assignments from your terminal, use the ASSIGN command.

You can assign tape drives directly using the following format:

ASSIGN MT*pdn* [*options*]

where *pdn* is the physical device number for the drive you want to assign. Use of certain options (such as –DENSITY) may require operator intervention.

You may request that the system operator assign a drive by using the MTX argument with the –ALIAS option. The format is

ASSIGN MTX –ALIAS MT*ldn* [*options*]

ASSIGN MTX asks the operator to assign any available drive that meets your specifications. *ldn* is a **logical device number** that you choose to refer to the drive in future operations. This may be different from the physical device number of the device actually assigned. Other options that you specify (if any) may instruct the operator to assign a drive that can handle a particular type of tape (for example, a 9-track tape at 6250 bpi). Options may also make special requests to the operator, for example, to remove the write-ring or to mount a particular tape.

Other ASSIGN Options

The –WAIT option indicates that you are willing to wait until the requested drive is available.

For a summary of all ASSIGN options, type HELP ASSIGN. For full information on using the ASSIGN command, see the *Data Backup and Recovery Guide*.

ASSIGN Related Messages

Suppose you request device MT3 with the command

```
OK, ASSIGN MT3
```

If your request is successful you receive the message

```
Device MT3 Assigned
OK,
```

If you request any available device, using, for example

```
OK, ASSIGN MTX -ALIAS MT0
```

you receive a message like the following:

```
Device MT1 Assigned
OK,
```

The device number (`MT1`) is the physical device number of the drive assigned to you. In future commands, you can use either this number or the logical device number that you chose with the –ALIAS option (`MT0`) to refer to the drive.

If you ask for a device that is in use and do not specify the –WAIT option, you receive the following message:

```
The device is in use.   (ASSIGN)
ER!
```

If the operator is not available to handle requests, any attempt by a user to assign a magnetic tape drive results in this message:

```
No Magtape assignment permitted. (asnmt$)
```

If a request cannot be handled by the operator for any reason, the following message appears at the terminal:

```
Magtape assignment request aborted (asnmt$)
```

The STATUS DEVICE Command

The STATUS DEVICE command allows you to see which physical devices (tape drives) are currently in use. The command displays physical and logical device numbers for any assigned magnetic tape drives. Tape drives not assigned are not shown in the display. The format for the STATUS DEVICE command is

STATUS DEVICE

The following example shows a typical display:

```
OK, STATUS DEVICE

Device User name                                  Usrnum Ldevice
MT0    DICKENS                                     13    MT0
MT1    POE                                         69    MT2
OK,
```

The display headings have the following meanings:

Heading	*Meaning*
`Device`	The physical device number
`User name`	The login name of the user who has the device assigned
`Usrnum`	The user number of that user
`Ldevice`	The logical device number being used for the drive

Releasing Tape Drives

When you complete a magnetic tape operation, release the magnetic tape drive for general use. Use the UNASSIGN command with one of the indicated arguments:

$$\text{UNASSIGN} \left\{ \begin{array}{l} \text{MT}\textit{pdn} \\ \text{–ALIAS MT}\textit{ldn} \end{array} \right\} \text{[–UNLOAD]}$$

If you requested a drive using the format ASSIGN MTX –ALIAS MT*ldn*, you can use the logical device number you chose:

UNASSIGN –ALIAS MT*ldn*

Otherwise, use the physical device number:

UNASSIGN MT*pdn*

In the following example you request a specific device:

```
OK, ASSIGN MT0

Device MT0 assigned.
OK,
.
.
.
OK, UNASSIGN MT0
```

In the following example, you request any available device:

```
OK, ASSIGN MTX -ALIAS MT1

Device MT2 assigned.
OK,
.
.
.
OK, UNASSIGN -ALIAS MT1
```

or you can use

```
OK, UNASSIGN MT2
```

Note

The –UNLOAD option normally rewinds the tape and places it offline. However, on some drives or controllers, –UNLOAD rewinds the tape but does not place it offline. Check with your System Administrator for the action provided by –UNLOAD at your installation.

Who Can Unassign a Drive?

A tape drive can be unassigned only by

- The user who assigned it (on default-privileged systems)
- The system operator

If an operator unassigns your tape drive, no message appears at your terminal. If you subsequently attempt to unassign the same device, an error message is displayed.

Backing Up and Restoring Files

After you have assigned a tape drive, use the MAGSAV and MAGRST commands to back up or restore files.

Backing Up Files With MAGSAV

You can back up files to the assigned tape drive by giving the MAGSAV command. The format is

MAGSAV [*options*]

Most backups do not require any options. Options for carrying out special operations such as selecting a special tape format are documented in the *Data Backup and Recovery Guide*.

After you invoke MAGSAV, the MAGSAV subsystem responds with a series of questions. The basic elements of the dialog are the following:

Prompt	*Response*
`Tape unit:`	Enter the physical or logical device number of an assigned drive (0-7).
`Enter logical tape number:`	For most saves, enter 1. See the *Data Backup and Recovery Guide* for information about specifying more than one logical tape.

Note

On machines that use cartridge tapes, you see the prompt `Overwrite or Append (o/a):` instead of `Enter logical tape number:`. Reply `O` for a new tape or to overwrite the contents of an old tape. Reply `A` to append backup to current tape contents.

`Tape name:`	Enter a tape name with a maximum of six characters.
`Date (MM DD YY):`	Press Return and PRIMOS enters the date.
`Rev no:`	Enter a decimal integer. You usually enter the revision of PRIMOS running on your system, for example, 22. If you press Return only, the revision number is set to 0.
`Name or command:`	Enter the save operation you want MAGSAV to perform. The most common responses are

```
filename
*
$A pathname
$R
```

For a complete list of responses, see the *Data Backup and Recovery Guide*.

filename causes MAGSAV to save the specified file or subdirectory from the current directory to tape.

`*` causes MAGSAV to save all files and directories in the current directory to tape.

`$A` *`pathname`* attaches you to the specified directory.

`$R` terminates the logical tape, rewinds the physical tape, and returns you to PRIMOS.

Each time you enter a name or command, MAGSAV attempts to carry out the specified operation and then prompts you for another name or command. If MAGSAV cannot carry out an operation, it displays an error message and then prompts for a new name or command. When you have saved all the files you want to save, enter $R to quit and rewind the tape. Remember to unassign the tape drive when you finish.

Restoring Files With MAGRST

Use the MAGRST command to restore files from tape to disk. The format is

MAGRST [*options*]

As with MAGSAV, you can carry out most restores without supplying any options. Options for special operations are documented in the *Data Backup and Recovery Guide.*

After you invoke MAGRST, the MAGRST subsystem responds with a series of questions.

Prompt	*Response*
`Tape unit (9 trk):`	Enter the physical or logical device number of an assigned drive (0-7).
`Enter logical tape number:`	For a single logical tape, enter 1. For information about restores from multiple logical tapes, see the *Data Backup and Recovery Guide.*
`Ready to restore:`	Enter the restore operation you want MAGRST to perform. The most common responses are

```
YES
NW [filename]
$A pathname
PARTIAL
```

For a complete list of responses, see the *Data Backup and Recovery Guide.*

`YES` restores the entire tape and returns you to PRIMOS. MAGRST restores objects to the directory to which you are currently attached. A directory tree on tape becomes a subdirectory tree of your current attach point.

`NO` requests a different tape unit and logical tape.

`NW` [*`filename`*] displays a tape index at your terminal. If you specify *filename*, the index is written to the specified file.

`$A` *`pathname`* attaches you to the specified directory.

`PARTIAL` restores specific files. After you specify `PARTIAL`, MAGRST prompts you for the pathnames of the files to restore, as follows.

Tree name:

Supply the pathname of a file or directory on the tape. MAGRST restores the file or directory using its objectname (the last component of the pathname) to the directory to which you are currently attached. For example, if you are attached to MYDIR and request that MAGRST restore the directory OLDIR>SUBDIR from tape, MAGRST copies SUBDIR to disk as a subdirectory of MYDIR, with the pathname MYDIR>SUBDIR.

Note

If you restore two files or directories with the same objectname to the same directory, the second overwrites the first.

MAGRST prompts you for a maximum of ten pathnames. MAGRST restores the specified files when you press Return alone. If you want to restore more than ten objects, repeat the restore procedure.

Be sure to unassign the tape drive when you finish all backup and restore operations.

Backup and Restore Examples

The following example shows how to assign and unassign drives and carry out simple backup and restore operations. For more complex operations, refer to the *Data Backup and Recovery Guide*.

In the first example, you save a file to tape:

```
OK, ASSIGN MT0
Device MT0 assigned.
OK, LD

<CPOSR3>MOZART (ALL access)
35 records in this directory, 43 total records out of quota of 0.

4 Files.

LOGIN.CPL         LOGIN.ABBREV      MY.GVARS          RESUME

4 Directories.

OPERAS            SYMPHONIES        QUARTETS          CONCERTOS

OK, MAGSAV
[MAGSAV Rev. 22.0 Copyright (c) 1988, Prime Computer, Inc.]
Tape unit (9 Trk): 0
Enter logical tape number: 1
Tape name: HITS
Date (MM DD YY): 3-11-88
Return
```

```
Rev no: 22
Name or Command: $I
Name or Command: RESUME
*** Start of Save ***
RESUME (dam)
*** End of Save ***

Name or Command: $R
1 Recovered MT IO errors.
OK, UNASSIGN MT0
```

In the following example, you later restore file RESUME to disk.

```
OK, ATTACH OLDIES
OK, LD

<CPOSR3>OLDIES (ALL access)
12 records in this directory, 22 total records out of quota of 0.

1 File.

MEMO

3 Directories.

OPERAS           SYMPHONIES       QUARTETS

OK, ASSIGN MT1
Device MT1 assigned.
OK, MAGRST
[MAGRST Rev. 22.0 Copyright (c) 1987, Prime Computer, Inc.]
You are not attached to an MFD.
Tape unit (9 Trk): 1
Enter logical tape number: 1
Name: HITS
Date(MM DD YY): 03-11-88
Rev no:       22
Reel no:       1
Ready to Restore: PARTIAL
Tree name: RESUME
Tree name:
Return
*** Starting Restore ***
*** End Logical tape ***
*** Restore Complete ***
```

```
OK, LD

<CPOSR3>OLDIES (ALL access)
12 records in this directory, 22 total records out of quota of 0.

2 Files.

RESUME           MEMO

4 Directories.

OPERAS           SYMPHONIES       QUARTETS

OK, UNASSIGN MT1
Device MT1 Released.
OK,
```

Other Magnetic Tape Operations

Prime supplies a variety of utilities for carrying out other magnetic tape operations.

ARCHIVE	Allows you to archive files from disk to tape and restore them from tape to disk. Automatically generates an online catalog of archived files.
TRANSPORT	Allows you to transfer files, via magnetic tape, from one Prime installation to another.
MAGNET	Allows you to transfer files, via magnetic tape, between Prime systems and other vendors' systems.

Consult the *Data Backup and Recovery Guide* for more information.

19 PRIMENET

Prime systems are often connected in **networks** that allow users to share the resources of many computers.

- Prime **local area networks, RINGNET** and **LAN300,** can connect several nearby Prime systems and user terminals.
- Point-to-point connections can tie distant Prime systems and local area networks together over dedicated communication lines.
- Prime systems can access both Prime and other vendors' computers connected to **Packet Switched Data Networks** (PSDNs) like TELENET™.

PRIMENET is a system of software and hardware that gives you easy access to all the resources available on your network. PRIMENET handles all of your work on the network in a **transparent** manner; you normally don't need to know about the physical connections among networked systems, and you often don't even need to know which system you are accessing.

This chapter introduces the basic PRIMENET user facilities. (For a complete discussion see the *User's Guide to Prime Network Services.*)

Note

The discussion that follows often refers to local and remote systems. Except for LAN300 users, the **local system** is the one your terminal is connected to (either directly or over a dialup line). If you are a LAN300 user, you can consider your local system to be the one to which you connect when you give the NTS CONNECT command. **Remote systems** are other systems on your network. (There can be both a local and several remote systems on a local area network.) Whenever you make use of a remote system, PRIMENET handles the communication.

You make use of PRIMENET with four different facilities:

Facility	*Description*
Remote File Access	While you are logged in to one system, you can access many of the files on other systems in your network exactly as if they were on your own system.
Remote Login	You can log in to any system on your network if you have a login ID.

NETLINK	You can use this utility to access both Prime and other vendors' systems connected over PSDNs. You can also use NETLINK over all Prime network types to supplement the capablilties of Remote File Access and Remote Login.
File Transfer Service (FTS)	You can use this separately priced product to transfer files between Prime systems in a network. FTS adds significantly to the capabilities available with the standard Remote File Access facility.

The capabilities of these facilities overlap to some extent. For example, you can transfer files to and from remote systems using FTS, but in many cases you can also simply copy them using Remote File Access. The following discussion gives you some guidelines about which system to use whenever capabilites overlap in this way.

Remote File Access

Remote File Access is the most transparent of the PRIMENET facilities. Your System Administrator can make certain disks from other systems on your network visible to your local system. You can access file system objects on these disks exactly as if they were on your own system; you just use their pathnames.

Making disks visible to your system in this way is called **adding disks**. Adding a disk doesn't affect the physical connection between the disk and the computer. It simply means that you can work with a disk physically connected to a remote machine as if it were connected to a local machine. For example, suppose your local system is connected in a network that includes several remote systems. Your System Administrator has chosen to add four disks located on these remote systems so that they are accessible to you via Remote File Access: <ACCT1> and <ACCT2> on the system SYSE, <PAYRL> on the system SYSC, and <GAMES> on the system MIS2. (Names like SYSE and MIS2 are established to identify each of the machines on a network when the network is configured. They are called **systemnames** or **nodenames**.)

You can attach to <ACCT2> just as if it were on your local system, with the command

```
OK, ATTACH <ACCT2>MFD
```

You can then list the contents of <ACCT2> with

```
OK, LD
```

You also get the listing without first attaching:

```
OK, LD <ACCT2>@@
```

You can work with any of the file system objects contained in these directories exactly as you would if they were on your own system. For example, you can edit the file <PAYRL>MONTHLY>AUGUST with the command

```
OK, ED <PAYRL>MONTHLY>AUGUST
```

Note

You cannot initiate a phantom process on your local system while you are attached to a remote disk. If you attempt this, you receive the error message `Illegal remote reference.`

As in the case of files on your local machine, you don't need to give the disk name when you give a pathname. You can, for example, just give the command

```
OK, ED MONTHLY>AUGUST
```

However, remember how PRIMOS locates a top-level directory (in this case the top-level directory is MONTHLY) when you omit the disk name. PRIMOS searches each disk beginning with logical device number 0 and attaches you to the first top-level directory that matches the specified name. When remote disks are added, PRIMOS searches all of the local disks first, and then searches remote disks in the order in which they were configured on your system. (You can discover this order with the STATUS DISKS command explained below.)

You can, therefore, safely omit the disk name for a remote file system object as long as you are sure that the top-level directory name does not occur on any other local or added remote disk. When top-level directory names are not unique, PRIMOS may or may not find the right one, depending on the order in which it searches the disks. If you are not sure, you are better off using the disk name in your pathnames. Using the disk name in remote pathnames can also give you significantly faster access, because otherwise PRIMOS may have to search through many disks to find the one you specify.

The STATUS DISKS Command

Use the **STATUS DISKS** command to find out which disks are available to you for Remote (as well as local) File Access and the order in which PRIMOS searches them. STATUS DISKS shows the names and logical device numbers of both local and remote disks. It also shows you the names of the systems on which remote disks are located. For example,

```
OK, STATUS DISKS

Disk    Ldev  Pdev  System

STATS     0    3462
BOOKS     1     460
MISCEL    2   71063
OUTPUT    3   71061
ACCT2     4          SYSE
ACCT1     5          SYSE
```

In the STATUS DISKS display, `Disk` is the name of the disk, `Ldev` is the logical device number, and `System` is the systemname. The disks that have *no* systemname listed are on the local system. (`Pdev` is the physical device number, which is normally not of concern to you.)

This display shows four disks on the local system as well as the four remote disks that the System Administrator has added. Remember that the STATUS DISKS listing gives you the order in which disks are searched for top-level directories when you don't specify the disk name in a pathname. In this case, the first disk to be searched is <STATS> (logical device 0), and the last is <GAMES> (logical device 7).

Suppose, for example, that both ACCT1 and ACCT2 contain top-level directories called PAYABLE. You can attach to <ACCT2>PAYABLE with the command

```
OK, A PAYABLE
```

because ACCT2 (logical device 4) is searched before ACCT1 (logical device 5). On the other hand, to attach to <ACCT1>PAYABLE, you must give the whole pathname:

```
OK, A <ACCT1>PAYABLE
```

ACLs on Remote Systems

Your ability to work with file system objects on remote systems is still governed by ACL protection, if ACL protection has been implemented for the remote system. For example, you must have U access to a remote directory in order to attach to it. Note that X access to an EPF on a remote system does not give you the right to execute the EPF. You need R access to execute a remote EPF.

Copying Remote File System Objects

Users frequently want to copy files or directories from remote systems to their local directories. When the file system object that you want is on a disk that has been added to your system, you can do this with the COPY command.

For example, you can copy the file <GAMES>BOARD>CHESS on system MIS2 to the directory <MISCEL>MYGAMES on your local system with the following command:

```
OK, COPY <GAMES>BOARD>CHESS <MISCEL>MYGAMES>CHESS
```

Remember, to copy an ACL protected file system object, you need R access to the object.

When a file system object is on a remote disk that has not been added to your system, you cannot copy it with the COPY command. In such a case, you may want to use FTS or NETLINK to copy the file. These procedures are explained in the sections on NETLINK and FTS below.

Adding Remote IDs

On some networks, the Network Administrator may restrict access to the disks on certain systems. In such cases, your System Administrator may add disks from those systems so that they are visible on your system. However, you still are not allowed Remote File Access to a disk on a restricted system unless you also have a valid user ID on the restricted system. This is called forced user validation.

Forced user validation protects the security of data on certain systems by limiting access privileges. Those users who have a valid ID on a restricted system can have Remote File Access to disks on that system. Those who don't have an ID on the system in question can not have Remote File Access to its disks.

Even if you have a valid user ID on a remote system that forces user validation, you still need to carry out one step in order to enable remote access to files on that system. You must issue the ADD_REMOTE_ID command at some point during your terminal session. This is called **adding a remote ID**. The command's format is

> ADD_REMOTE_ID ***remote-ID*** [***password***] –ON ***systemname***
> [–PROMPT] [–PROJECT ***project-ID***]

The arguments and options for this command are the same as those for the LOGIN command (explained below in the section Remote Login), with one exception. With the –PROMPT option, the command prompts you for the password instead of forcing you to enter it on the command line.

remote-ID is a valid user ID that you have on the remote system specified by *systemname*. You must supply any *password* or *project-ID* required for access to the remote system.

In essence, the ADD_REMOTE_ID command makes a list of user IDs that PRIMENET can use when it attempts to access files on various restricted remote systems. You may have remote IDs for as many as 16 different systems simultaneously, but only one for any given remote system. For example, if you add the remote ID EARTH on SYSA and then later add the remote ID AIR on SYSA, AIR replaces EARTH.

Note

The ADD_REMOTE_ID command does not actually establish a user ID for you on the remote system. Only the System Administrator of the remote system can do this. When you try to access a remote file on a restricted system, PRIMENET must supply a valid user ID to that system. PRIMENET does this automatically for you. ADD_REMOTE_ID simply tells PRIMENET which ID and passwords to use when it accesses a specific remote system for you.

As an example of the use of ADD_REMOTE_ID, assume that your local system is SYSP, but that you also have a user ID, WATER, on SYSK on your network. Your System Administrator has added several disks from the system SYSK so that you can have Remote File Access to them, but SYSK is a restricted system that forces user validation for Remote File Access. You can access the added disks, but first you must issue the ADD_REMOTE_ID command:

```
OK, ADD_REMOTE_ID WATER PASS -ON SYSK -PROJECT POETRY
```

WATER is now available with the password PASS as a remote ID for accessing SYSK. This ID remains available for the duration of your terminal session or until you remove it (with REMOVE_REMOTE_ID, as described below).

Suppose, for example, that one of the added disks from SYSK is called <SONNET>. You can now access the files on <SONNET> just as if they were on your own system. For example, you can attach to the directory <SONNET>SHAKESPEARE with the following command:

```
OK, ATTACH <SONNET>SHAKESPEARE
```

You receive an error message if

- You have not previously issued the ADD_REMOTE_ID command
- You have added the wrong ID
- You do not have a valid ID on SYSK

```
OK, ATTACH <SONNET>SHAKESPEARE
Slave validation error.  MFD (ATTACH)
ER!
```

If you have a different user ID on a remote system, you may find it useful to add that remote ID even if the system does not restrict file access. For example, your remote ID may have greater access rights than your local ID to files on the remote system. If you add the remote ID, you gain the remote ID's access rights when you access files on the remote system.

For example, suppose the ACL protecting the directory <MUSIC>BAROQUE on remote system SYSM is

```
.MGROUP   ALL
$REST     LUR
```

Suppose your user ID PIANO on SYSM is included in the group .MGROUP, while your user ID FLUTE on your local system is not. If you access a file in <MUSIC>BAROQUE from your local system without first adding a remote ID, you have the access rights accorded to FLUTE. In this case, FLUTE is a member of $REST and has only LUR rights. If you add the remote ID PIANO, you get ALL access rights as a member of .MGROUP.

Because a remote ID is valid only for the duration of the login session, you may find it convenient to incorporate the ADD_REMOTE_ID command in your LOGIN.CPL file if you frequently access remote disks on restricted systems. If you do this, use the ADD_REMOTE_ID –PROMPT option. When –PROMPT is specified, ADD_REMOTE_ID asks for your password on the remote system when the LOGIN.CPL executes. Thus, your password on the remote system does not need to appear in the LOGIN.CPL file.

Examining Your Remote IDs

Use the LIST_REMOTE_ID command to examine the remote IDs you have currently established. The command format is

LIST_REMOTE_ID [**–ON** *systemname*]

If you give the –ON option, only the remote ID for *systemname* is listed; if you omit the –ON option, all of your remote IDs are displayed. Passwords are never displayed. For example,

```
OK, LIST_REMOTE_ID
System  User ID                          Project id
------  -------                          ----------
SYSB       EARTH
SYSK     WATER                              POETRY
SYSM      FIRE
OK,
```

Removing Remote IDs

Your remote ID list can contain a maximum of 16 remote IDs, one ID per system. If your list has reached the 16-ID limit, you cannot add more remote IDs unless you remove at least one with the REMOVE_REMOTE_ID command. (To list existing remote IDs, use the LIST_REMOTE_ID command, explained above.)

The command format for REMOVE_REMOTE_ID is

REMOVE_REMOTE_ID –ON ***systemname***

where *systemname* is the nodename of the system whose ID is to be deleted. If *systemname* is not in the list, you receive the error message **Not found**.

Remote Login

You can have login IDs on several machines in a network. If you have a login ID on a remote machine, you can log in to that machine using the –ON option of the LOGIN command. The format is

LOGIN ***user-id*** **[*login-password*] –ON** ***systemname*** **[–PROJECT** ***project-id*****]**

where *systemname* is the name of the system you want to log in to.

For security reasons, your System Administrator may disallow passwords on the login line. In this case, PRIMOS prompts you for a password. If any of the systems on which you work uses specific project names, you must supply the appropriate *project-id* when you log in. Your administrator gives you any required project IDs when you receive your user ID.

You may be able to log in to one system on your network while you are still logged in to another. This depends on how the System Administrator has configured the system you are currently logged in to. When you log in to the second system, PRIMOS automatically logs you out of the first.

When you log in to a remote system, PRIMENET establishes a connection to route input and output between your terminal and the remote system. In fact, your terminal remains physically connected to your local system, but PRIMENET routes your terminal input and output so that you can work as if you had a direct connection to the remote system. This connection is called a **virtual circuit**. If you log out or fail to log in successfully, the connection is broken. Your terminal is then left connected to your local machine, but not logged in.

The following example shows remote login and logout.

```
OK, LOGIN MYSELF -ON MIS2

MYID(user 33) logged out Wednesday, 28 Oct 87 15:45:36.
Time used: 06h 56m connect, 03m 24s CPU, 00m 21s I/O.
PRIMENET 22.0.0 MIS2
Password?                                    You type the wrong password.
Invalid user ID or password; please try again.

Disconnected from MIS2                       You are connected to your local
OK, LOGIN MYSELF -ON MIS2                        machine, but not logged in.
PRIMENET 22.0.0 MIS2
Password?                                     You type the right password.

MYSELF(user 6) logged in Wednesday, 28 Oct 15:46:28.
Welcome to Primos version 22.0.0
Copyright(c) 1987 Prime Computer Inc.
OK,
 .                                            You do your work on MIS2.
 .
 .
OK, LO

MYSELF(user 6) logged out Wednesday, 28 Oct 87 15:55:09.
Time used: 00h 9m connect, 00m 04s CPU, 00m 21s I/O.

Wait...

Disconnected from MIS2
OK,                           You are connected to local machine but not logged in.
```

Network Status

You can find the names and states of all systems (or nodes) in the network by giving the following command:

STATUS NETWORK

The following example shows the state of an eight-system network as it is displayed for a local user on the SYSA system. The UP state means that the connection to the remote system is working. The system to which you have logged in is listed first and shown by asterisks (****).

```
OK, STATUS NETWORK

Ring Network

    Node       State
    SYSA       ****
    SYSE        Up
    SYSB        Up
    SYSC        Up
    MIS2        Up
    MIS1       Down
    ENG3        Up
    SYSW       Down

OK,
```

When to Use Remote Login

You can log in remotely only to a system on which you have a valid user ID. Even if you do have a valid user ID, remember that you may be able to use file resources on a remote system via Remote File Access instead of logging in. You need to log in remotely when the disks on a remote system have not been added to your local system and are thus not available via Remote File Access.

You can also use remote login as an alternative to Remote File Access when your user ID on the remote system has greater access rights than your user ID on the local system. Logging in to the remote system allows you the greater access rights of your remote ID. (In these circumstances you can also add the remote ID using the ADD_REMOTE_ID command, as discussed above in the section, Adding Remote IDs.)

Remote login is also useful when you do a lot of work on a remote system. You can have a different LOGIN.CPL, abbreviation set, prompts, and other custom features for each login ID. If you do a different kind of work on each system, you can customize your user environment differently for each user ID.

Using NETLINK for Remote Access

The NETLINK facility allows you to connect both to Prime systems on your network and to Prime and other vendors' systems over a public data network.

NETLINK allows you to

- Transfer text files across networks
- Set data transmission characteristics
- Display the status of your connection
- Connect to and use a maximum of six different remote systems at the same time
- Specify the various fields of the connect packet when data transmission characteristics of another vendor's system differ from those of your local system

Only NETLINK's basic usage is presented in this section. For a list of all NETLINK commands and error messages, see the *User's Guide to Prime Network Services*.

Basic NETLINK Usage

The basic steps to using NETLINK are

1. Enter NETLINK command mode by issuing the NETLINK command. The format is

   NETLINK

   When you enter command mode, the NETLINK @ prompt appears.
2. Connect to the remote system by issuing the C subcommand. The format is

   C *address*

   address is either the host address assigned by the public data network or a PRIMENET system nodename.

   When a connection has been established, the following message appears:

   ```
   address Connected
   PRIMENET Rev 22.0.0 systemname
   ```

   Log in to the system as you would normally, entering any passwords, as required.
3. Once you finish a terminal session, log out as you would normally. The following message appears:

   ```
   address Disconnected
   @
   ```

   When a connection to a remote host has been terminated by logging out, command mode is reentered and the @ appears. You may now connect to another site or return to PRIMOS on your local system. To return to PRIMOS on your local system, enter the following command:

   ```
   @ QUIT
   ```

NETLINK Example

The following example illustrates a basic terminal session.

```
OK, NETLINK
[NETLINK  Rev. 22.0]

@  C SYS9

SYS9 Connected
PRIMENET 22.0 SYS9
LOGIN HOBBITT
Password?                                  Password is not echoed on terminal.
```

```
HOBBITT (user 28) logged in Friday, 07 Dec 88 10:45:32.
Welcome to PRIMOS version 22.0
Last login Friday, 07 Dec 88 10:38:28.

OK,
            .
            .
            .

OK, LOGOUT

HOBBITT (user 28) logged out Friday, 07 Dec 88 10:46:44.
Time used: 00h 01m connect, 00m 01s CPU, 00m 01s I/O.

Wait...

SYS9 Disconnected

@ QUIT

OK,
```

Alternate NETLINK Usage: The –TO Option

You may use the following option on the NETLINK command line:

–TO *address*

This bypasses NETLINK's @ prompt, and connects you directly to *address*. You can then log in, do your work, and log out. After logging out, you return directly to your original system, without getting the @ prompt or needing to give the QUIT command.

For example,

```
OK, NETLINK -TO SYS9
[NETLINK  Rev. 22.0]

SYS9 Connected
PRIMENET 22.0 SYS9
LOGIN HOBBITT
Password?                          Password is not echoed on terminal.

HOBBITT (user 28) logged in Friday, 07 Dec 88 10:57:56.
Welcome to PRIMOS version 22.0
Last login Friday, 07 Dec 88 10:55:36.

OK,
             .
             .
             .

OK, LOGOUT
```

```
HOBBITT (user 28) logged out Friday, 07 Dec 88 10:58:08.
Time used: 00h 00m connect, 00m 01s CPU, 00m 00s I/O.

Wait...

SYS9 Disconnected

OK,
```

The –MODE REMOTE_ECHO Option

If you use ECL or the EMACS screen editor remotely, you may also wish to use the –MODE REMOTE_ECHO option to the NETLINK command. This option enables you to receive screen echoes faster and makes work at the keyboard easier.

You enter the option in one of two places:

- On the NETLINK command line. For example,

  ```
  OK, NETLINK -TO SYSZ -MODE REMOTE_ECHO
  ```

- Following the @ prompt. For example,

  ```
  @ C SYSZ -MODE REMOTE_ECHO
  ```

This option is explained more fully in the *User's Guide to Prime Network Services.*

The NETLINK HELP Facility

You can get a brief summary of NETLINK subcommands and options by typing HELP in response to the @ prompt. For example,

```
OK, NETLINK
[NETLINK Rev. 19.3.2]

@ HELP
```

Following the HELP display, you receive another @ prompt and can continue with your NETLINK session. More information on the NETLINK HELP facility appears in the *User's Guide to Prime Network Services.*

When to Use NETLINK

Whenever you want to connect to a public data network, you must use NETLINK. Whether you use NETLINK to connect to systems on your own network depends on whether you need NETLINK's extra capabilities. You can, of course, log in to remote systems on your network via remote login without using NETLINK. However, you can only use remote login with one system at a time. Even if your local system permits you to log in remotely while you are still logged in locally, you are automatically logged out of your local system.

When you use NETLINK, you can connect to another system and still remain logged in to the local system. This means that you can connect to a remote system, do your work, and return to the local system without having to log in again. NETLINK also allows you to establish as many as six connections with other systems simultaneously. (See the *User's Guide to Prime Network Services* for details.)

Finally, you may find it easier to use EMACS and ECL over the network if you use the NETLINK –MODE REMOTE_ECHO option.

Transferring Files Between Systems With FTS

The FTR command provides a method of transferring files between Prime systems that are connected via PRIMENET links. The FTR command is part of the File Transfer Service (FTS), which is a separately priced product.

To transfer a file, you submit a request that details all the necessary information for the transfer to occur. You can make a request even when the communications link between the two systems is not operational or the remote system is down, because requests are queued on the local (requesting) system.

Once you have submitted a request for a file transfer, you may display or cancel the request. The following sections explain these operations briefly. Further information is available in the *PRIMOS Commands Reference Guide* and in the *User's Guide to Prime Network Services.*

Access Rights

In order for full FTS to work correctly, you need the following access rights for your user ID:

Access	*Function*
LUWR to the source directory	Sending files
ALURW to the destination directory	Receiving files

In addition, the FTS Server needs access rights. The FTS Server is a special phantom that performs your FTS work for you. The server needs the following rights:

Access	*Directory*
DALURW	Directory containing source file
DALURW	Directory to receive file
U	All higher directories in source and destination pathnames

The user ID of the FTS Server is set by your System Administrator. Find out from your Administrator the user ID of your system's FTS Server so that you can grant the server appropriate access rights.

To keep your main directory private, you can create a special subdirectory for your transfers and give the FTS Server the necessary rights to this directory only.

Source and Destination Sites

File transfers take place between sites. A **site** (also called a system or node) is a single computer, identified by a unique sitename; Prime sites normally use their PRIMENET systemnames as sitenames. Files are transferred from a source site to a destination site. One of these must be your local site; the other is usually a remote site. (FTS cannot transfer files between two remote sites in a single step.)

The following discussion assumes that you are using FTS between Prime machines that have been configured using the FTGEN command, explained in the *PRIMENET Planning and Configuration Guide*. To transfer files to or from sites that are not configured, see the *User's Guide to Prime Network Services*.

Temporary Destination Files

When FTS transfers a file, the receiving system initially creates the destination file with a temporary filename. FTS prepends the letters T$ to the destination filename to create the temporary filename. The name may be truncated to ensure that it does not exceed 32 characters. To display the temporary filename enter the LD command during the transfer.

Progress on creating and renaming the file is recorded in the user's log file. On successful completion of the transfer, FTS attaches to the destination directory, deletes any existing file with a name that matches the destination filename specified by the user, and renames the temporary file, giving it the requested destination filename.

Request Names and Request Numbers

Each request has two means of identification associated with it: a request name and a unique request number. FTS assigns the number when you submit the request. You can use either the name or the number to identify the request. The name is either the name of the file to be transferred or a specific name that you assign to the request when you submit it. You usually refer to the request by its name. You may use the unique request number to distinguish between two requests that have the same name.

Sending a File

To send a file to another Prime computer, use the FTR command with the following format:

FTR ***source-pathname destination-pathname*** **–D**STN_**S**ITE ***sitename***

source-pathname is the pathname of the file to be sent. You may give just a filename if the file is in your current directory.

destination-pathname specifies the name to give the file at the destination site after it has first been successfully transferred to a temporary file. Directories specified in the destination pathname must already exist for the transfer to work. FTS does not create directories.

Note

If the pathname for a source file or destination file requires directory passwords, include the passwords in the pathname. Put the whole pathname within single quotation marks. For example,

```
'MARPLE CLUE>EVIDENCE'
```

–DSTN_SITE *sitename* specifies the name of the destination site.

FTR displays the following response to your request:

```
Request request-name (request-number) submitted.
```

request-number is the unique identification number assigned by FTR. *request-name* is the source filename.

For example, assume you are on SYS2. The following example shows an FTR send request:

```
OK, FTR LINDEN>SQUARES.FTN ELM>SQUARES.FTN -DS SYS9
[FTR Rev.5.0 Copyright (c) 1986, Prime Computer, Inc.]
Request SQUARES.FTN (177777) submitted.
OK,
```

In this example, FTR queues a copy of the file SQUARES.FTN in the directory LINDEN to send to system SYS9, for deposit in the directory ELM under the name SQUARES.FTN. This file is initially transferred to the temporary file T$SQUARES.FTN and on successful completion of the transfer is renamed to SQUARES.FTN. The request name is the source filename, SQUARES.FTN. The unique request number is 177777. Single quotation marks are required around the pathname containing the password.

Obtaining a File

To get a file from another Prime computer, use the FTR command in the following format:

FTR ***source-pathname destination-pathname*** **–SRC_SITE** ***sitename***

source-pathname and *destination-pathname* are used as defined above in the section, Sending a File.

–SRC_SITE *sitename* specifies the name of the site where the desired file is stored.

For example, assume you are on SYS2. The command

```
OK, FTR PEOPLE>LIST MYDIR>MYLIST -SRC_SITE SYS6
```

copies the file PEOPLE>LIST on SYS6 to the directory MYDIR on SYS2. The new copy of the file is MYLIST. The request name is the source filename, LIST.

Printing a File at a Remote Site

To print a hard copy of a file on a printer at another site, use the following format:

```
FTR source-pathname -DSTN_SITE sitename -DEVICE LP -DSTN_USER name
```

–DEVICE LP is the option that instructs FTS to print the file on a line printer at the remote site specified by –DSTN_SITE *sitename*.

–DSTN_USER *name* specifies the name of the person to receive the printout at the remote site. The file is printed with the name of the FTS Server (set by your System Administrator) on the first line of the banner, where your user ID usually appears. The *name* specified after –DSTN_USER appears on the second line of the banner of the printed file. *name* need not be a user ID.

For example, assume you are on SYSA. The command

```
OK, FTR STUART>LETTER -DSTN_SITE SYSF -DEVICE LP -DSTN_USER JUDY_JONES
```

causes the file LETTER to be printed (at a line printer) on SYSF. The first line of the banner is the name of the FTS server; the second line is JUDY_JONES. The request name is LETTER.

Deferring the Transfer of a File

You can delay the transfer of a file by specifying the –DEFER option followed by *date-time* when you submit your request. –DEFER has the following format:

```
FTR -DEFER date-time
```

date-time is in the format *yy-mm-dd.hh:mm:ss*. If you only wish to defer the file's transfer to a time later in the same day, you can leave out the year, month, and day and just specify the time in 24 hour format. For example,

```
OK, FTR PEOPLE>LIST MYDIR>MYLIST -SRC_SITE SYS6 -DEFER 18:00
```

This file would be transferred at 6:00 p.m. on the same day the request was made.

Setting the Relative Priority of Your Request

You can set the relative priority for your transfer request with the –PRIORITY option. The System Administrator sets the upper and lower limits for priorities, which normally range from 1 (lowest) through 7 (highest). The default value is usually 5. For example, if you wish to transfer a file with highest priority, specify

```
OK, FTR MYDIR>LETTER YOURDIR>LETTER -DSTN_SITE SYS2 -PRIORITY 7
```

FTS does not process a lower priority request until it has processed requests submitted with a higher priority. It is possible for lower priority requests to remain in the queue for long periods of time if other users are continually submitting higher priority requests.

The System Administrator can disable the –PRIORITY option or restrict access to some priorities to allow for urgent requests. When the –PRIORITY option is disabled, FTS ignores priority settings and services the queue in the order in which users submit requests.

Checking the Status of Requests

Use the –STATUS and –DISPLAY options of the FTR command to check the status of your file transfer requests.

Using the –STATUS Option: Once you have submitted a request you can check its status with the following command:

$$\text{FTR -STATUS} \left[\begin{Bmatrix} \textit{request-name} \\ \textit{request-number} \end{Bmatrix} \right]$$

The –STATUS option returns a brief, one-line report for each of your file transfer requests identified by *request-name* or *request-number*. If you omit the request name or number, you receive a report on all of your current requests. The report has the following form. (Certain items are not displayed unless they pertain to the current request.)

```
date.time user-id request-name(request-#)(Queue-#(priority))Status -category
```

date.time is in the form *yy-mm-dd.hh:mm:ss*. Status *-category* is one of the following:

Category	*Meaning*
waiting	The request is in the transfer queue, but has not yet been transferred.
transferring	The transfer is in progress.
put on hold by user **put on hold by operator** **put on hold by FTS**	A request is being retained in the transfer queue. (The *User's Guide to Prime Network Services* explains how to hold files.)
aborting	An active transfer request cannot be executed.

If a defer symbol (D) follows the status category of a request, a defer time is supplied with that request.

For example,

```
OK, FTR -STATUS
[FTR Rev. 5.0 Copyright (c) 1986, Prime Computer, Inc.]
87-02-10.14:31:04 LINDEN SQUARES.FTN (177777) (QU$I(5))
                    Status - transferring
OK,
```

In this example, LINDEN has one request, named SQUARES.FTN, with a priority of 5. It is being transferred.

Using the –DISPLAY Option: You can ask for a full report on the status of your requests, with the following command:

$$\textbf{FTR –DISPLAY} \left[\begin{Bmatrix} \textit{request-name} \\ \textit{request-number} \end{Bmatrix} \right]$$

Specifying *request-name* displays full information on all current requests with this name. Specifying *request-number* displays full information on the request with this number. If you omit *request-name* and *request-number*, FTR displays detailed information on all of your current requests.

The display takes the following form. (Items are displayed only if they pertain to the current request.)

Category	*Information on the Request*
`Request`	Request name (request number).
`User`	User ID of submitter.
`Queue`	Queue name where the request is queued.
`Queued`	Date and time the request was submitted, and the status of the request; Status may be `waiting`, `transferring`, or `put on hold` by either the user or the operator.
`Priority`	The relative priority of the transfer request.
`Last attempt`	Date and time of the most recent transfer attempt, and the number of transfer attempts.
`Current time`	Current date and time.
`Defer time`	The date and time the request is to be processed.
`Source file`	Source pathname.
`Source file size`	Number of bytes; displayed only if the source file is on the local site.
`Destination file`	Destination pathname.
`Source site`	Source site name.
`Destination site`	Destination site name.
`Request log file`	Pathname of log file; not always displayed. FTR must have Write access to create a log file.
`Log message level`	Level of detail entered in the request log field.
`Source user`	A user ID (or another name) at the source site to be associated with the transferred file; not always displayed; useful when notifying a user at a remote site about a transfer.
`Source file type`	Type of file being transferred.
`Destination file type`	Type of destination file.
`Destination user`	A user ID (or another name) at the destination site to be associated with the transferred file; not always displayed; useful when printing files at remote sites or when notifying a user at a remote site about a transfer.
`Options`	List of active options.

For example, assume your user ID is LINDEN on SYS7:

```
OK, FTR SQUARES.FTN ELM>SQUARES.FTN -DU OAK -DS SYS9
-LOG LINDEN>LOG -MSGL TRACE -DEFER 88-12-12.11:00:00
[FTR Rev.5.0 Copyright (c) 1986, Prime Computer, Inc.]
Request SQUARES.FTN (177777) submitted.
OK, FTR -DISPLAY
[FTR Rev.5.0 Copyright (c) 1986, Prime Computer, Inc.]
Request            - SQUARES.FTN (177777)
User               - LINDEN
Queue              - QU$1
Queued             - 88-12-12.10:25:57  Status    - waiting
Priority           - 5
Last attempt       - 00-00-00.00:00:00  Attempts -      0
Current time       - 88-12-12.10:26:05
Defer time         - 88-12-12.11:00:00
Source file        - <FOREST>LINDEN>SQUARES.FTN
Source file size   -          86 bytes.
Destination file   - ELM>SQUARES.FTN
Source site        - SYS7
Destination site   - SYS9
Request log file   - <FOREST>LINDEN>LOG
Log message level  - TRACE
Source user        - LINDEN
Destination user   - OAK
Source file type            - SAM
Destination file type       - SAM
Options :-
BINARY, COPY, NO DELETE
OK,
```

Logging Request Events

You can create an automatic log of file transfer request events by specifying the –LOG option, in the format

–LOG ***pathname***

when you submit a request. FTR deposits logging information, including information regarding the progress of creating and renaming the temporary file, in *pathname* on the system originating the request. If *pathname* already exists, the logging information is appended to the end of the file.

For example,

```
OK, FTR SALLY>INFO WILLIAM>INFORM -DS SYSG -LOG SALLY>FTR.LOG
```

If the transfer of INFO is successful, the entries in the log file FTR.LOG look like this:

```
15.55.13: [1.1] Request INFO (954232) started Friday, December 5,
1988.
15.55.14: [1.1] Submitting user is SALLY.
15.55.14: [1.1] Local file is <DISKZ>SALLY>INFO.
15.55.14: [1.1] Temporary file is <DISKZ>SALLY>T$INFO.
15.55.30: [1.1] RESULT: Transfer Terminated: Satisfactory and
Complete.
15.55.30: [1.1] File successfully renamed.
15.55.30: [1.1] Request INFO (954232) finished.
```

You can increase the degree of detail entered in your log file if you wish. Specify on the FTR command line the –MESSAGE_LEVEL option with one of the following arguments: DETAILED, STATISTICS, TRACE. You must also specify the –LOG option. More information on the –MESSAGE_LEVEL option appears in the *User's Guide to Prime Network Services*.

Requesting Notification About Transfers

You may request that notification about the progress of a submitted request be sent to a source and/or a destination user. Use the notify options, given below, both for transferring and fetching operations.

Option	*Description*
–SRC_NTFY	Notifies the source user when the file transfer starts and ends. In the case of a receive, you must include the following option to inform FTR whom to notify.
–SRC_USER ***user-ID***	(Required with –SRC_NTFY fetch.)
–DSTN_NTFY	Notifies the destination user when the file transfer starts and ends. In the case of a send, you must include the following option to tell FTR whom to notify.
–DSTN_USER ***user-ID***	(Required with –DSTN_NTFY send.)

With the following command line, user LINDEN on SYS2 invokes FTR to transfer file SQUARES.FTN to ELM>RESULTS.FTN on SYS9, and requests that notification be sent to both LINDEN and ELM:

```
OK, FTR SQUARES.FTN ELM>RESULTS.FTN -DS SYS9 -SN -DN -DU ELM
```

In the next example, user LINDEN on SYS2 fetches file SUM.FTN from ELM on SYS9, names it RESULTS.FTN, and requests that notification be sent to both LINDEN and ELM:

```
OK, FTR ELM>SUM.FTN RESULTS.FTN -SS SYS9 -DN -SN -SU LINDEN
```

Note

The options –SRC_NTFY and –DSTN_NTFY use the PRIMOS message facility to tell users of a successful transfer. However, if the specified user is not logged in or has set MESSAGE status to –REJECT, then notification is not received. Using the –LOG option instead of, or in addition to, –SRC_NTFY and –DSTN_NTFY provides a permanent record of the operation.

Canceling Requests

If you have submitted a request that is currently waiting in a queue to be transferred, you may cancel the request with the following command:

$$\textbf{FTR -CANCEL} \begin{Bmatrix} \textit{request-name} \\ \textit{request-number} \end{Bmatrix}$$

For example, if the request you want to cancel is named NEWS and the *request-number* is 5684210, either of the following commands cancels the request:

```
FTR -CANCEL NEWS
FTR -CANCEL 5684210
```

You receive the following message:

```
Request NEWS (5684210) cancelled.
```

You cannot cancel requests that are in the process of being transferred.

The FTR Help Facility

You can request a brief summary of FTR options at the terminal by giving the FTR command without arguments or options:

FTR

For a list of subjects on which you can get a full help display use the –HELP option alone:

FTR –HELP

You can get information on any subject with the format

FTR –HELP ***subject***

Requests on Hold

Under certain circumstances, FTS automatically puts a request on hold. If you receive a message to this effect or an FTR –DISPLAY screen shows that your request is on hold, you may wish to do one of the following:

- Give the command

  FTR –CANCEL $\left\{\begin{array}{l}\textit{request-name}\\ \textit{request-number}\end{array}\right\}$

  This command cancels the request, and you can resubmit it.
- Give the command

  FTR –RELEASE $\left\{\begin{array}{l}\textit{request-name}\\ \textit{request-number}\end{array}\right\}$

  This command instructs FTR to try the transfer again. Try to determine what caused the failure, and resubmit a corrected request.

If you requested that a log file be generated, you can look at the log to find out why the file is being held. Either you or the operator can hold a file intentionally with the –HOLD option. More information on this feature is given in the *User's Guide to Prime Network Services.*

Other Options

The FTR command has other options that allow you to modify, abort, and otherwise control your requests. Full information on these options (as well as on the –HOLD and –RELEASE options) appears in the *PRIMENET Guide.*

When to Use FTR

If you want to copy a file from a remote disk that has been added to your system, you can often do so simply by using the COPY command. If the file you want is on a disk that has not been added to your system, or if it is on a system that forces user validation on which you don't have a valid user ID, you can use FTS. Remember, though, that you must have sufficient ACL rights to the file or destination directory. FTS also automates the file transfer process. If, for example, the remote system is down, FTS can automatically retry your request later.

20 The Condition Mechanism

PRIMOS has a **condition mechanism**, which is activated when any executing process encounters certain unusual events. These events (or conditions) fall into one of three categories:

- Software-puzzling situations: illegal addresses, end of file encountered while reading data, and so on
- Hardware and arithmetic exceptions: numbers too large or too small for the computer to handle, attempts to divide by zero, program too large for its allotted space, and so on
- External occurrences: situations not directly controlled by the executing process, such as the use of the BREAK key from the user's terminal

PRIMOS defines many such conditions. Some examples are

Condition	*Definition*
`ACCESS_VIOLATION$`	Process has attempted to read, write, or execute in a segment to which it has no access.
`ARITH$`	Arithmetic exception.
`STACK_OVF$`	Process has overflowed its stack segment.
`QUIT$`	User has typed Break or Control P at terminal.
`ILLEGAL_INST$`	Process has tried to execute an illegal instruction.
`ENDFILE (file)`	End of file encountered while a PL/I file was read.

For a complete list of these conditions, see Appendix D. Full information on the condition mechanism appears in the *Subroutines Reference III: Operating System.*

Using the Condition Mechanism

The condition mechanism's goal is either to repair the problem and restart the program or to terminate the program in an orderly manner. To achieve this goal, the condition mechanism activates diagnostic or remedial subroutines (or PL/I begin blocks) called **on-units**.

Users writing in FORTRAN IV, FORTRAN 77, PL/I, or PMA can define their own on-units within the procedures for which they are intended. However, all users are automatically protected by PRIMOS system on-units. When an error condition occurs, the condition mechanism looks for on-units within the executing procedure. If it finds none or if the procedure's on-units call for further help, the condition mechanism searches first through any calling procedure's on-units and then through the system's on-units, activating the first appropriate on-unit it finds.

The System Default On-unit

Of all the system on-units, the system default on-unit is the one you are most likely to encounter. This on-unit displays a message in the following format and then returns you to PRIMOS command level:

```
Error:  condition condition raised at address
        additional information
```

You may then take any one of the following actions:

- Give the START command. The condition mechanism tries to resume running the program from the point at which the condition was raised.
- Give the DUMP_STACK command. This command prints a stack dump, which traces the sequence of calls and returns by which the program reached its current state. The stack dump can be displayed at the terminal or written to a file, as you prefer. If you are familiar with Prime machine architecture, you may find that the DUMP_STACK command gives you enough information to solve your problem. (For details, see the *PRIMOS Commands Reference Guide.*) You may START a program again after dumping the stack.
- Run the program under DBG. (The program must be compiled with the –DEBUG option.) If the DUMP_STACK command did not provide enough information to solve the problem, this is probably the best action to take.
- Give the RELEASE_LEVEL command to release the errant program. You remain at PRIMOS command level and can give any PRIMOS command you choose.

Note

If you invoke the system default on-unit for a process running as a phantom or Batch job, the condition mechanism writes the error message into the job's command output file and then logs out the process.

Writing On-units

You can write your own on-units. These can be tailored to the programs and procedures for which they are written. For more information, see the *Subroutines Reference III: Operating System.*

Appendices

A

Glossary

This glossary defines the most important terms and concepts mentioned in the *PRIMOS User's Guide.*

absolute pathname
A pathname that begins with a diskname.

access category
A file system object that contains only an ACL. An access category can protect any number of file system objects in the same directory.

access control list
See ACL.

access right
A user's right to carry out a specific operation on a file system object. For example, a user with Read (R) access to a file can read the file.

ACL
A list of users and their access rights, also called an access control list. When you establish an ACL for a file system object, the ACL controls access to the object.

active
Designates an EPF that has been suspended without running to completion and remains mapped to the user's address space. You can often restart an active EPF with the START command.

active window
Specifies the time span when a batch queue is active. Jobs in a batch queue are only executed during the queue's active window.

adding disks
Making disks on remote systems available for remote file access.

address space
Memory area available to a user or process.

addressing mode
Specifies how a program calculates memory addresses. You must compile programs using an addressing mode compatible with the linking utility that you use to link them.

auto speed detect
Allows PRIMOS to detect the baud rate of a user's terminal automatically.

basename
The first component of an objectname. Objectnames often have two components, a basename and a suffix, separated by a period. For example, PROGRAM is the basename in the two-component objectname PROGRAM.BIN.

Batch processing
A system for executing programs without tying up user terminals. Programs submitted to the Batch system are placed in queues to await execution. Batch processing can control the priority, time allotted, and other characteristics of execution in order to use system resources efficiently.

Batch queue
A list of programs awaiting execution under control of the Batch system.

binary file
In general, a file whose contents cannot be interpreted as character data. In programming, an object file is often called a binary file.

buffer
An area of memory used for temporary storage of data. Buffers are frequently used to hold data being input from or output to a device.

character data
Data that consists of letters, numbers, and other symbols as well as control characters. Control characters are special characters that often cannot be displayed on a terminal screen or printed on a printer. Many control characters can be generated by pressing Ctrl in combination with another key at the keyboard. Control characters are often used to control display devices and printers.

COMI file
See command input file.

command argument
An element of a command line that specifies something to be acted upon by the command. Arguments are usually pathnames of file system objects or identifying names such as a user ID.

command environment
The part of PRIMOS responsible for interpreting commands, invoking and suspending the requested programs, and displaying prompts. The command environment maintains information about the status of each program invoked.

command environment depth
The maximum number of command levels allowed in a user's command environment.

command environment limits
Limits on the number of command levels, program invocations per command level, and memory segments available to a user.

command history
In ECL, a set of up to 200 previously entered command lines that remain available for editing and resubmittal.

command input file
A text file containing a sequence of PRIMOS commands. When you submit a command input file with the COMINPUT command, PRIMOS executes the sequence of commands. Also called a COMI file.

command level
PRIMOS and interactive subsystems are at command level when they are able to accept user commands. PRIMOS and subsystems usually display prompts to indicate that they are at command level. In PRIMOS, command level also refers to the number of programs a user has simultaneously suspended. When no programs are suspended, the user is said to be at command level 1. With each suspended program, the user's command level increases by 1.

command line
A string consisting of commands, arguments, and options, terminated by a Return, that a user submits to PRIMOS or a subsystem.

command output file
See COMO file.

command procedure language
See CPL.

command processor
The part of PRIMOS that interprets command lines.

COMO file
A file that records terminal input and output. Also called a command output file.

compiler
A utility that converts a program source file written in a specific programming language into an object file.

condition mechanism
A part of PRIMOS that handles errors and other interruptions to executing programs.

CPL
A language for writing programs consisting of PRIMOS commands and CPL directives. CPL directives control the flow of command execution, pass command line arguments, and carry out other functions typical of high-level programming languages. Also called Command Procedure Language.

current attach point
The directory in which PRIMOS locates a user for the purpose of file system references. PRIMOS interprets objectnames and relative pathnames as beginning with the current attach point.

current directory
See current attach point.

current line
When editing a file with the ED text editor, the current line is the line of the file to which the edit pointer currently points. In INPUT mode, text is added at the current line.

DBG
The Prime Source Level Debugger. To use the debugger, you must compile your programs with the –DEBUG option.

debugging
The process of finding and correcting programming errors. You can use the DBG utility to aid you in debugging programs.

default
A value or setting that is used unless the user explicitly specifies something else.

default protection
ACL protection that a file system object receives from the directory that contains the object. A file system object is default protected unless you explicitly protect it with a specific ACL or access category.

delimiter
In ED, a character used to mark the beginning or end of a literal string in a command line.

destination
A name used to specify a printer to the SPOOL command. Use the destination as an argument to the –ATTRIBUTE option.

device
A mechanical device used for input and output, such as a tape drive, disk drive, or terminal.

directive
In CPL, a statement used to control program flow, pass command line arguments, and carry out other programming functions. CPL directives begin with &.

directory
A file system object that contains a list of names of other file system objects, information about their characteristics (such as date and time last modified), and information that PRIMOS can use to locate them. In effect, a directory functions as an index to a group of file system objects.

directory tree
The hierarchy of subdirectories and other file system objects below a given directory in the PRIMOS file system.

diskname
A name that identifies a logical disk in file system operations. A diskname is the first element of a full pathname. In the pathname, the diskname appears between angle-brackets in the format <*diskname*>. For example, in the pathname <MUSIC>SONATAS>MOZART, MUSIC is the diskname.

disk number
The logical device number of a disk. When PRIMOS searches disks for a top-level directory, it searches in disk number order.

disk volume
See disk.

disk quota
Limit on the amount of disk space that a directory can occupy.

disk
In PRIMOS, disk, partition, logical disk, and disk volume are all used to refer to a set of mechanical disk surfaces that PRIMOS treats as a single logical unit. Disk also refers to a physical platter in a magnetic disk storage pack, but PRIMOS users work with logical rather than physical disks.

dynamic runfile
See EPF.

echo
The display of characters on the terminal screen as they are typed at the keyboard.

ECL
The command line editor. Allows users to edit, save, and resubmit command lines in PRIMOS and in some subsystems. Editing features include searching, deleting, copying, and moving of command line text.

ED
A line oriented text editor. Used for creating and modifying text files.

empty line
In ED, a line consisting of a single carriage return. Used for switching between INPUT and EDIT modes.

EPF
A runfile created with the BIND linking utility. PRIMOS can map EPFs to any available memory space, so that several EPFs can be maintained in a user's memory space at once. EPF names use the .RUN suffix. EPFs are also called executable program formats or dynamic runfiles.

erase character
A character that deletes the preceding character from a command line when the command line is processed. The default erase character for PRIMOS and many subsystems is the double quotation mark (").

error prompt
A prompt displayed by PRIMOS or a subsystem when a command does not execute to completion. `ER!` is the default PRIMOS error prompt.

executable file
See runfile.

executable program format
See EPF.

file
An organized collection of data, stored on disk or another peripheral storage medium. Each file is identified by a filename. For PRIMOS users, the file is the basic unit of organization of data in the file system.

file system
The logical organization of data stored on disk. The PRIMOS file system is hierarchically organized. PRIMOS users can manipulate information using the logical organization of the file system and need not concern themselves with the physical locations of data.

file system object
A file, directory, access category, or segment directory in the PRIMOS file system.

file transfer service
See FTS.

file-unit
A channel through which all input to and output from a file takes place. File-units are identified by octal numbers.

filename
A name that identifies a file in the PRIMOS file system. Filenames can have from 1 to 16 components, separated by periods. Filenames often have two components: a basename that identifies the file's contents, and a standard suffix that identifies the file's function. For example, COUNT.C identifies the source code of a C language program called COUNT, and COUNT.RUN identifies the executable file.

forced user validation
To require validation of a user's ID on a remote system for remote file access. If you want to use remote file access on a remote system that forces user validation, you must have a valid user ID on the remote system, and you must identify that ID to your local system with the ADD_REMOTE_ID command.

form name
A name used to specify a paper type to the SPOOL command. Use the form name as and argument to the –ATTRIBUTE option.

FTS
A utility for transferring files between Prime systems connected via PRIMENET. FTS queues transfer requests and processes them automatically so that you can continue to work or even log off while you wait for the transfer.

full pathname
See ordinary pathname.

function
A routine that returns a value. Command functions are functions that you can call from a PRIMOS command line using the format [*function-name*]. The PRIMOS command processor substitutes the returned value for the function call when it processes the command line.

global variable
In PRIMOS, variables that take their values from a global variable file. Once they are defined in a global variable file, global variables are accessible from PRIMOS command level, in abbreviations, to CPL programs, to command input files, and to programs in some high-level languages.

group name
A name for a group of user IDs. Group names begin with a period (.), for example, .STAFF. You can use a group name in an ACL to grant the same rights to all members of a group.

IAP
See initial attach point.

initial attach point
The directory in the file system to which you are attached when you log in. Also called the origin directory.

interactive system
A system that carries out a dialog with terminal users. Users supply commands or other input from the terminal. Many interactive systems display prompts to indicate that they are ready to accept input.

invoke
To initiate the execution of a program.

iteration list
A list of two or more arguments for a command. The list is enclosed in parentheses and the arguments are separated by spaces. PRIMOS executes the command for each argument in turn.

key field
Defines the location of a data item to a file-handling utility.

kill character
A character that deletes all preceding characters in a command line when PRIMOS processes the command line. The default kill character for PRIMOS and many subsystems is the question mark (?).

LAN
See local area network.

LAN300
A Prime proprietary local area network.

library
A set of subroutines that many programs can call.

linking utility
A utility that creates an executable file from the object file created by a language compiler.

literal character
A special character treated as ordinary text. For example, in ED the double quotation mark (") is normally a delete character. You can force ED to treat the double quotation mark as a literal character, by preceding it with the escape character, caret (^). When you do this, ED treats the double quotation mark as an ordinary text character rather than as a delete character.

local area network
A network connecting a number of nearby systems over special lines.

local system
In a network, the computer to which a user's terminal is connected. If you use LAN300, your local system is the one to which you connect with the NTS CONNECT command.

logical device number
An octal number by which PRIMOS identifies a device. A device's logical device number may be unrelated to the device's actual physical address or physical device number.

logical disk
See disk.

login file
A program or command file PRIMOS executes each time you log in. Login filenames are LOGIN.RUN, LOGIN.SAVE, LOGIN.CPL, or LOGIN.COMI.

login-password
A password that PRIMOS uses to validate your right to log in under a given user ID.

long prompt
A PRIMOS prompt that displays time, usage, and command level information. You can invoke long prompts with the –LONG option of the RDY command.

command macro
In ECL, a series of ECL editing commands that you can save and later invoke with a single key combination.

manage-option
Option of the JOB command used to control the execution of a Batch job after it is submitted.

mapped
Copied to a user's address space. An EPF is mapped to your address space when you invoke it for the first time. The EPF remains mapped for some time after it completes execution so that you can quickly reinvoke it.

master file directory
See MFD.

MFD
The highest level directory in the PRIMOS file system. An MFD lists the contents of a disk. The objectname for all MFDs is MFD, and the pathname for the MFD of a given disk is <*diskname*>MFD.

mini-command level
Results when a user has reached the maximum number of command levels allowed by the user's command environment (called command environment depth). At mini-command level, PRIMOS accepts only a limited set of commands.

monitor-option
Option of the JOB command used to monitor the status of a Batch job after it is submitted.

name generation
Creating new objectnames based on other objectnames in a command line. You use the name generation characters, = and ^, to show where elements of the old names should be substituted in the new names.

NETLINK
Software that allows you to connect both to Prime systems on your local network and Prime and other vendors' systems over a public data network.

network
A system of hardware and software that connects computers and allows them to communicate.

Network Terminal Service
See NTS.

nodename
A name that identifies a computer in a network.

not active
An EPF that has run to completion but remains mapped to a user's address space is said to be not active.

NTS
Software that controls links between terminals and computers in a LAN300 network.

null line
In ED, a line that contains no text. ED inserts null lines at various points in the memory copy of a file. New text lines are added at the null lines. Null lines do not become part of the file when it is saved on disk.

object file
A generalized version of a program, created by a language compiler from the source code. You must link the object file with a linking utility to create an executable file that the computer can run.

objectname
A name that identifies a file system object. An objectname can have a maximum of 16 components, separated by periods. Many objectnames have two components: a basename that identifies the contents of the object, and a standard suffix that indicates the object's function. An objectname is the last element of a pathname.

octal number
A number in base 8, shown with a subscript 8, for example 12_8.

on-units
Routines that handle errors encountered during program execution. The system default on-unit displays an error message describing the problem encountered.

operating system
The program that organizes all work on the computer system. The operating system provides the user interface, shares system resources among processes, and manages hardware operations. PRIMOS is the operating system for 50 Series computers.

option
An optional term that modifies the action of a command. Option names always begin with a hyphen (–).

ordinary pathname
A pathname that begins with the name of a top-level directory.

origin directory
See initial attach point.

parent directory
The directory that contains a file system object is the object's parent directory.

partition
See disk.

pathname
A series of directory names, ending in an objectname, that describes a path through a file system tree structure to a file system object. Angle-brackets separate the directory names in a pathname. For example, the pathname AUTHORS>MYSTERY>POE>RAVEN, locates the file RAVEN in a directory tree beginning with the directory AUTHORS.

phantom process
A process that executes disconnected from a terminal.

Prime ECS
See Prime Extended Character Set.

Prime Extended Character Set
The Prime extended version of the ANSI ASCII 7 bit character set. The Prime ASCII character set includes all standard 7 bit ASCII characters with the eighth bit set to 1, so that PRIME characters in the range decimal 128 to 255 correspond to standard ASCII characters in the range 0 to 127. In the Prime Extended Character Set, characters with the eighth bit set to 0 are also significant. Appendix C lists the complete extended character set. Also called Prime ECS.

PRIMOS
The operating system for all 50 Series computers.

priority ACL
An ACL, set by the operator, that overrides all other ACLs set for objects on a disk. Operators set priority ACLs so that they can back up disks regardless of the ACL protection set on individual files.

process
The basic unit that PRIMOS uses to organize its work. PRIMOS establishes a process for each logged-in user, for each phantom, and for certain other system functions. Each process operates in its own environment, including such characteristics as a specific terminal line assignment (for user processes), attach point, and the like. A major organizational task of PRIMOS is to share the system's physical resources among processes.

project
A group of user IDs, identified by a project ID. Every user ID belongs to a project. If your user ID has not been assigned a default project, you need to supply the project ID when you log in.

prompt
Text that PRIMOS and interactive subsystems display to indicate that they are ready to accept user input.

ready prompt
A PRIMOS prompt that indicates that the last command was successfully executed and that PRIMOS is ready to accept another command. `OK,` is the default PRIMOS ready prompt.

receive state
Controls the flow of messages sent with the MESSAGE command to a user terminal.

relative pathname
A pathname specified relative to a user's current attach point. Relative pathnames begin with *>. For example, the pathname *>SCULPTURE>RODIN, describes a path from the current attach point, through the subdirectory SCULPTURE, to the file RODIN.

remote file access
Use of files located on remote systems as if the files were on your local system.

remote ID
A user ID on a remote system.

remote login
Login to a remote system from a terminal connected to the local system.

remote system
In a network, a system tc which the user's terminal is not directly connected. If you use LAN300, systems other than the one you connect to with the NTS CONNECT command are remote systems.

$REST
In an ACL, $REST designates all users not otherwise specified by usernames or groupnames.

return
To calculate a value. Used for functions and subroutines.

RINGNET
A Prime local area network in which computers are connected in a ring configuration. Each user terminal is connected directly to one computer on the ring.

runfile
A program in a form that can be executed by the computer. You use a linking utility to create a runfile from an object file. Also called an executable file.

site
In FTS, refers to a computer that is the source or destination of a file transfer. Sitenames identify sites.

source code
A text file containing a program written in a programming language. You must compile source code with a compiler and link the resulting object file with a linking utility to create an executable file.

specific ACL
An ACL that defines access to a single file system object. A specific ACL is an attribute of the object it protects, and only exists as long as the object exists.

spool
To submit a file to a spool queue for printing. You spool files with the SPOOL command.

spool queue
A list of files waiting to be printed.

static runfile
An executable file that must always be loaded into the same area of user memory.

string
A series of characters treated as text. A string may contain numerals, for example, but the numerals are treated simply as elements of text rather than as numerical values.

subdirectory
A directory contained within another directory.

submit-option
Option of the JOB command used when a Batch job is submitted for Batch execution.

subsystem
An interactive system that has its own commands and prompts.

suffix
The last component of an objectname with two or more components. Suffixes usually indicate the function of the file system object. PRIMOS and subsystems recognize a number of standard suffixes. For example, the .RUN suffix identifies an EPF.

syntax suppression
Prevents the PRIMOS command interpreter from processing some elements of the command line. For example, syntax suppression prevents PRIMOS from evaluating global variables in a command line. Instead, the variables are treated as literal strings. A tilde character (~) at the beginning of a command line invokes syntax suppression.

system level
PRIMOS command level.

system prompt
One of the prompts displayed at PRIMOS command level.

systemname
See nodename.

text file
A file consisting of character data.

text formatter
A program that processes text files containing special formatting codes to produce formatted output that can be displayed at a terminal or printed on a printer.

top-level directory
The first level of subdirectory within an MFD. System Administrators often assign a top-level directory to each user.

tree structure
The hierarchical structure of the PRIMOS file system. In the file system, directories can contain other directories, which can in turn contain other directories, and so on. The result is that the file system branches below each MFD like an inverted tree.

treewalking
Using wildcard characters in an intermediate position in a pathname to make a command act on designated objects throughout a file system tree structure.

unmapped
Refers to an EPF that is available to a user in the file system, but is not mapped to the user's address space.

user ID
A name by which PRIMOS identifies a user logged in to the system. You must have a valid user ID to log in.

utility
A program that carries out a specialized set of operations. For example, programs that carry out linking operations are called linking utilities.

variable
A symbolic name for a quantity that can have more than one actual value.

virtual circuit
A connection made via PRIMENET between two networked systems. A virtual circuit behaves as if it were a direct connection even though the actual connection is routed through the network.

warning prompt
A prompt that PRIMOS displays when a command executes to completion but generates a warning condition while executing. The default PRIMOS warning prompt is the same as the ready prompt, `OK,`.

wide area network
A network connecting computers over long distances using public communication lines.

wildcard character
A character that can be substituted for one or more characters or components of an objectname in a command line. PRIMOS searches a directory for actual objectnames that match the pattern of the wildcard characters, and substitutes them in the command line. You use wildcards to make a single command line act on a number of file system objects.

B Systems Defaults and Constants

Prime systems have the following defaults and constants.

Terminal Defaults

Full duplex

X-ON/X-OFF disabled

Buffered protocol (using carrier detect) disabled

Input error-checking disabled

PRIMOS Keyboard Standards

Character	*Octal Value*	*Interpretation*
Ctrl P	220_8	BREAK (Interrupt)
Return	215_8	Newline, converted to 212_8 (linefeed) by standard software

PRIMOS Keyboard Defaults

Character	*Interpretation*
"	Character erase
?	Line kill

Note

You can change your erase and kill characters with the TERM command.

PRIMOS Command Line Standards

Character	*Interpretation*
+	Wildcard for 1 character *or* add a literal for name generation
@	Wildcard for 1 or more characters
^	Wildcard negation *or* exclusion for name generation
=	Name generation
;	Command separator
~	Command processor syntax suppressor

Protection

File System Objects in ACL-protected Directories

Newly created files are opened with RW rights; once stored, they assume default protection (the protection existing on the directory where they reside). Other objects are created with default protection.

Files in Password-protected Directories

New files are created with the following protection:

Owner:	All access rights (RWD)
Non-owner:	No access rights (NIL)

C

The Prime Extended Character Set

As of Rev. 21.0, Prime has expanded its character set. The basic character set remains the same; it is the ANSI ASCII 7-bit set (called ASCII-7) with the 8th bit turned on. However, the 8th bit is now significant; when it is turned off, it signifies a different character. Thus, the size of the character set has doubled from 128 to 256 characters. This expanded character set is called the **Prime Extended Character Set (Prime ECS).**

The pre-Rev. 21.0 character set is a proper subset of Prime ECS. These characters have not changed. Software written before Rev. 21.0 continues to run exactly as it did before. Software written at Rev. 21.0 or later that does not use the new characters needs no special coding to use the old ones.

Prime ECS support is automatic at Rev. 21.0 or later. You may begin to use characters that have the 8th bit turned off. However, the extra characters are not available on most printers and terminals. Check with your System Administrator to find out whether you can take advantage of the new characters in Prime ECS.

Table C-1 shows the Prime Extended Character Set. The pre-Rev. 21.0 character set consists of the characters with decimal values 128 through 255 (octal values 200_8 through 377_8). (The pre-Rev. 21.0 character set is shaded in Table C-1.) The characters added at Rev. 21.0 all have decimal values less than 128 (octal values less than 200_8).

Specifying Prime ECS Characters

Direct Entry

On terminals that support Prime ECS, you can enter the characters directly; the characters appear on the screen as you type them. For information on how to do this, see the appropriate manual for your terminal.

A terminal supports Prime ECS if

- It uses ASCII-8 as its internal character set.
- The TTY8 protocol is configured on your asynchronous line.

If you do not know whether your terminal supports Prime ECS, ask your System Administrator.

On terminals that do not support Prime ECS, you can enter any of the ASCII-7 printing characters (characters with a decimal value of 160 or higher) directly by just typing them.

Octal Notation

If you use the Editor (ED), you can enter any Prime ECS character on any terminal by typing

^*octal-value*

where *octal-value* is the three-digit octal number given in Table C-1. You must type all three digits, including leading zeros.

Before you use this method to enter any of the ECS characters that have decimal values between 32 and 127, first specify the following ED command, which permits ED to print as ^*nnn* any characters that have a first bit of 0.

```
MODE CKPAR
```

Character String Notation

The way in which you specify Prime ECS characters in character strings in programs depends on the characters you wish to specify and the programming language used. For rules describing how to specify Prime ECS characters in character strings, refer to the appropriate language manual.

Special Meanings of Prime ECS Characters

Either PRIMOS or an applications program running on PRIMOS may interpret some Prime ECS characters in a special way. For example, PRIMOS interprets Ctrl P (220_8) as a process interrupt. ED, the Editor, interprets the backslash (\) as a logical tab. If you wish to make use of the Prime ECS backslash character in a file you are editing with ED, you must define another character as your logical tab. For a detailed description of how PRIMOS interprets special characters, see Chapter 1.

Prime Extended Character Set Table

Table C-1 contains all of the Prime ECS characters, arranged in ascending order. This order shows both the collating sequence and the way that comparisons are done for character strings. For each character, the table includes the graphic, the mnemonic, the description, and the binary, decimal, hexadecimal, and octal values. A blank entry indicates that the particular item does not apply to this character. The graphics for control characters are specified as ^*character*; for example, ^P represents the character produced when you type P while holding Ctrl down.

Characters with decimal values from 000 to 031 and from 128 to 159 are control characters. Characters with decimal values from 032 to 127 and from 160 to 255 are graphics characters.

TABLE C-1
The Prime Extended Character Set

Graphic	Mnemonic	Description	Binary	Decimal	Hex	Octal
	RES1	Reserved for future standardization	0000 0000	000	00	000
	RES2	Reserved for future standardization	0000 0001	001	01	001
	RES3	Reserved for future standardization	0000 0010	002	02	002
	RES4	Reserved for future standardization	0000 0011	003	03	003
	IND	Index	0000 0100	004	04	004
	NEL	Next line	0000 0101	005	05	005
	SSA	Start of selected area	0000 0110	006	06	006
	ESA	End of selected area	0000 0111	007	07	007
	HTS	Horizontal tabulation set	0000 1000	008	08	010
	HTJ	Horizontal tab with justify	0000 1001	009	09	011
	VTS	Vertical tabulation set	0000 1010	010	0A	012
	PLD	Partial line down	0000 1011	011	0B	013
	PLU	Partial line up	0000 1100	012	0C	014
	RI	Reverse index	0000 1101	013	0D	015
	SS2	Single shift 2	0000 1110	014	0E	016
	SS3	Single shift 3	0000 1111	015	0F	017
	DCS	Device control string	0001 0000	016	10	020
	PU1	Private use 1	0001 0001	017	11	021
	PU2	Private use 2	0001 0010	018	12	022
	STS	Set transmission state	0001 0011	019	13	023
	CCH	Cancel character	0001 0100	020	14	024
	MW	Message waiting	0001 0101	021	15	025
	SPA	Start of protected area	0001 0110	022	16	026
	EPA	End of protected area	0001 0111	023	17	027
	RES5	Reserved for future standardization	0001 1000	024	18	030
	RES6	Reserved for future standardization	0001 1001	025	19	031
	RES7	Reserved for future standardization	0001 1010	026	1A	032
	CSI	Control sequence introducer	0001 1011	027	1B	033
	ST	String terminator	0001 1100	028	1C	034
	OSC	Operating system command	0001 1101	029	1D	035
	PM	Privacy message	0001 1110	030	1E	036

TABLE C-1
The Prime Extended Character Set - Continued

Graphic	Mnemonic	Description	Binary	Decimal	Hex	Octal
	APC	Application program command	0001 1111	031	1F	037
	NBSP	No-break space	0010 0000	032	20	040
¡	INVE	Inverted exclamation mark	0010 0001	033	21	041
¢	CENT	Cent sign	0010 0010	034	22	042
£	PND	Pound sign	0010 0011	035	23	043
¤	CURR	Currency sign	0010 0100	036	24	044
¥	YEN	Yen sign	0010 0101	037	25	045
¦	BBAR	Broken bar	0010 0110	038	26	046
§	SECT	Section sign	0010 0111	039	27	047
¨	DIA	Diaeresis, umlaut	0010 1000	040	28	050
©	COPY	Copyright sign	0010 1001	041	29	051
ª	FOI	Feminine ordinal indicator	0010 1010	042	2A	052
«	LAQM	Left angle quotation mark	0010 1011	043	2B	053
¬	NOT	Not sign	0010 1100	044	2C	054
	SHY	Soft hyphen	0010 1101	045	2D	055
®	TM	Registered trademark sign	0010 1110	046	2E	056
¯	MACN	Macron	0010 1111	047	2F	057
°	DEGR	Degree sign	0011 0000	048	30	060
±	PLMI	Plus/minus sign	0011 0001	049	31	061
²	SPS2	Superscript two	0011 0010	050	32	062
³	SPS3	Superscript three	0011 0011	051	33	063
´	AAC	Acute accent	0011 0100	052	34	064
µ	LCMU	Lowercase Greek letter µ, micro sign	0011 0101	053	35	065
¶	PARA	Paragraph sign, Pilgrow sign	0011 0110	054	36	066
·	MIDD	Middle dot	0011 0111	055	37	067
¸	CED	Cedilla	0011 1000	056	38	070
¹	SPS1	Superscript one	0011 1001	057	39	071
º	MOI	Masculine ordinal indicator	0011 1010	058	3A	072
»	RAQM	Right angle quotation mark	0011 1011	059	3B	073
¼	FR14	Common fraction one-quarter	0011 1100	060	3C	074

TABLE C-1
The Prime Extended Character Set - Continued

Graphic	Mnemonic	Description	Binary	Decimal	Hex	Octal
½	FR12	Common fraction one-half	0011 1101	061	3D	075
¾	FR34	Common fraction three-quarters	0011 1110	062	3E	076
¿	INVQ	Inverted question mark	0011 1111	063	3F	077
À	UCAG	Uppercase A with grave accent	0100 0000	064	40	100
Á	UCAA	Uppercase A with acute accent	0100 0001	065	41	101
Â	UCAC	Uppercase A with circumflex	0100 0010	066	42	102
Ã	UCAT	Uppercase A with tilde	0100 0011	067	43	103
Ä	UCAD	Uppercase A with diaeresis	0100 0100	068	44	104
Å	UCAR	Uppercase A with ring above	0100 0101	069	45	105
Æ	UCAE	Uppercase diphthong Æ	0100 0110	070	46	106
Ç	UCCC	Uppercase C with cedilla	0100 0111	071	47	107
È	UCEG	Uppercase E with grave accent	0100 1000	072	48	110
É	UCEA	Uppercase E with acute accent	0100 1001	073	49	111
Ê	UCEC	Uppercase E with circumflex	0100 1010	074	4A	112
Ë	UCED	Uppercase E with diaeresis	0100 1011	075	4B	113
Ì	UCIG	Uppercase I with grave accent	0100 1100	076	4C	114
Í	UCIA	Uppercase I with acute accent	0100 1101	077	4D	115
Î	UCIC	Uppercase I with circumflex	0100 1110	078	4E	116
Ï	UCID	Uppercase I with diaeresis	0100 1111	079	4F	117
Đ	UETH	Uppercase Icelandic letter Eth	0101 0000	080	50	120
Ñ	UCNT	Uppercase N with tilde	0101 0001	081	51	121
Ò	UCOG	Uppercase O with grave accent	0101 0010	082	52	122
Ó	UCOA	Uppercase O with acute accent	0101 0011	083	53	123

TABLE C-1
The Prime Extended Character Set - Continued

Graphic	Mnemonic	Description	Binary	Decimal	Hex	Octal
Ô	UCOC	Uppercase O with circumflex	0101 0100	084	54	124
Õ	UCOT	Uppercase O with tilde	0101 0101	085	55	125
Ö	UCOD	Uppercase O with diaeresis	0101 0110	086	56	126
×	MULT	Multiplication sign used in mathematics	0101 0111	087	57	127
Ø	UCOO	Uppercase O with oblique line	0101 1000	088	58	130
Ù	UCUG	Uppercase U with grave accent	0101 1001	089	59	131
Ú	UCUA	Uppercase U with acute accent	0101 1010	090	5A	132
Û	UCUC	Uppercase U with circumflex	0101 1011	091	5B	133
Ü	UCUD	Uppercase U with diaeresis	0101 1100	092	5C	134
Ý	UCYA	Uppercase Y with acute accent	0101 1101	093	5D	135
Þ	UTHN	Uppercase Icelandic letter <u>Thorn</u>	0101 1110	094	5E	136
ß	LGSS	Lowercase German letter double <u>s</u>	0101 1111	095	5F	137
à	LCAG	Lowercase a with grave accent	0110 0000	096	60	140
á	LCAA	Lowercase a with acute accent	0110 0001	097	61	141
â	LCAC	Lowercase a with circumflex	0110 0010	098	62	142
ã	LCAT	Lowercase a with tilde	0110 0011	099	63	143
ä	LCAD	Lowercase a with diaeresis	0110 0100	100	64	144
å	LCAR	Lowercase a with ring above	0110 0101	101	65	145
æ	LCAE	Lowercase diphthong <u>ae</u>	0110 0110	102	66	146
ç	LCCC	Lowercase c with cedilla	0110 0111	103	67	147
è	LCEG	Lowercase e with grave accent	0110 1000	104	68	150
é	LCEA	Lowercase e with acute accent	0110 1001	105	69	151
ê	LCEC	Lowercase e with circumflex	0110 1010	106	6A	152

TABLE C-1
The Prime Extended Character Set - Continued

Graphic	Mnemonic	Description	Binary	Decimal	Hex	Octal
ë	LCED	Lowercase e with diaeresis	0110 1011	107	6B	153
ì	LCIG	Lowercase i with grave accent	0110 1100	108	6C	154
í	LCIA	Lowercase i with acute accent	0110 1101	109	6D	155
î	LCIC	Lowercase i with circumflex	0110 1110	110	6E	156
ï	LCID	Lowercase i with diaeresis	0110 1111	111	6F	157
ð	LETH	Lowercase Icelandic letter Eth	0111 0000	112	70	160
ñ	LCNT	Lowercase n with tilde	0111 0001	113	71	161
ò	LCOG	Lowercase o with grave accent	0111 0010	114	72	162
ó	LCOA	Lowercase o with acute accent	0111 0011	115	73	163
ô	LCOC	Lowercase o with circumflex	0111 0100	116	74	164
õ	LCOT	Lowercase o with tilde	0111 0101	117	75	165
ö	LCOD	Lowercase o with diaeresis	0111 0110	118	76	166
÷	DIV	Division sign used in mathematics	0111 0111	119	77	167
ø	LCOO	Lowercase o with oblique line	0111 1000	120	78	170
ù	LCUG	Lowercase u with grave accent	0111 1001	121	79	171
ú	LCUA	Lowercase u with acute accent	0111 1010	122	7A	172
û	LCUC	Lowercase u with circumflex	0111 1011	123	7B	173
ü	LCUD	Lowercase u with diaeresis	0111 1100	124	7C	174
ý	LCYA	Lowercase y with acute accent	0111 1101	125	7D	175
þ	LTHN	Lowercase Icelandic letter Thorn	0111 1110	126	7E	176
ÿ	LCYD	Lowercase y with diaeresis	0111 1111	127	7F	177

TABLE C-1
The Prime Extended Character Set - Continued

Graphic	Mnemonic	Description	Binary	Decimal	Hex	Octal
	NUL	Null	1000 0000	128	80	200
^A	SOH/TC1	Start of heading	1000 0001	129	81	201
^B	STX/TC2	Start of text	1000 0010	130	82	202
^C	ETX/TC3	End of text	1000 0011	131	83	203
^D	EOT/TC4	End of transmission	1000 0100	132	84	204
^E	ENQ/TC5	Enquiry	1000 0101	133	85	205
^F	ACK/TC6	Acknowledge	1000 0110	134	86	206
^G	BEL	Bell	1000 0111	135	87	207
^H	BS/FE0	Backspace	1000 1000	136	88	210
^I	HT/FE1	Horizontal tab	1000 1001	137	89	211
^J	LF/NL/FE2	Line feed	1000 1010	138	8A	212
^K	VT/FE3	Vertical tab	1000 1011	139	8B	213
^L	FF/FE4	Form feed	1000 1100	140	8C	214
^M	CR/FE5	Carriage return	1000 1101	141	8D	215
^N	SO/LS1	Shift out	1000 1110	142	8E	216
^O	SI/LS0	Shift in	1000 1111	143	8F	217
^P	DLE/TC7	Data link escape	1001 0000	144	90	220
^Q	DC1/XON	Device control 1	1001 0001	145	91	221
^R	DC2	Device control 2	1001 0010	146	92	222
^S	DC3/XOFF	Device control 3	1001 0011	147	93	223
^T	DC4	Device control 4	1001 0100	148	94	224
^U	NAK/TC8	Negative acknowledge	1001 0101	149	95	225
^V	SYN/TC9	Synchronous idle	1001 0110	150	96	226
^W	ETB/TC10	End of transmission block	1001 0111	151	97	227
^X	CAN	Cancel	1001 1000	152	98	230
^Y	EM	End of medium	1001 1001	153	99	231
^Z	SUB	Substitute	1001 1010	154	9A	232
^[	ESC	Escape	1001 1011	155	9B	233
^\	FS/IS4	File separator	1001 1100	156	9C	234
^]	GS/IS3	Group separator	1001 1101	157	9D	235
^^	RS/IS2	Record separator	1001 1110	158	9E	236
^_	US/IS1	Unit separator	1001 1111	159	9F	237
	SP	Space	1010 0000	160	A0	240
!		Exclamation mark	1010 0001	161	A1	241
''		Quotation mark	1010 0010	162	A2	242
#	NUMB	Number sign	1010 0011	163	A3	243
$	DOLR	Dollar sign	1010 0100	164	A4	244
%		Percent sign	1010 0101	165	A5	245
&		Ampersand	1010 0110	166	A6	246

TABLE C-1
The Prime Extended Character Set - Continued

Graphic	Mnemonic	Description	Binary	Decimal	Hex	Octal
'		Apostrophe	1010 0111	167	A7	247
(		Left parenthesis	1010 1000	168	A8	250
)		Right parenthesis	1010 1001	169	A9	251
*		Asterisk	1010 1010	170	AA	252
+		Plus sign	1010 1011	171	AB	253
,		Comma	1010 1100	172	AC	254
−		Minus sign	1010 1101	173	AD	255
.		Period	1010 1110	174	AE	256
/		Slash	1010 1111	175	AF	257
0		Zero	1011 0000	176	B0	260
1		One	1011 0001	177	B1	261
2		Two	1011 0010	178	B2	262
3		Three	1011 0011	179	B3	263
4		Four	1011 0100	180	B4	264
5		Five	1011 0101	181	B5	265
6		Six	1011 0110	182	B6	266
7		Seven	1011 0111	183	B7	267
8		Eight	1011 1000	184	B8	270
9		Nine	1011 1001	185	B9	271
:		Colon	1011 1010	186	BA	272
;		Semicolon	1011 1011	187	BB	273
<		Less than sign	1011 1100	188	BC	274
=		Equal sign	1011 1101	189	BD	275
>		Greater than sign	1011 1110	190	BE	276
?		Question mark	1011 1111	191	BF	277
@	AT	Commercial at sign	1100 0000	192	C0	300
A		Uppercase A	1100 0001	193	C1	301
B		Uppercase B	1100 0010	194	C2	302
C		Uppercase C	1100 0011	195	C3	303
D		Uppercase D	1100 0100	196	C4	304
E		Uppercase E	1100 0101	197	C5	305
F		Uppercase F	1100 0110	198	C6	306
G		Uppercase G	1100 0111	199	C7	307
H		Uppercase H	1100 1000	200	C8	310
I		Uppercase I	1100 1001	201	C9	311
J		Uppercase J	1100 1010	202	CA	312
K		Uppercase K	1100 1011	203	CB	313
L		Uppercase L	1100 1100	204	CC	314
M		Uppercase M	1100 1101	205	CD	315
N		Uppercase N	1100 1110	206	CE	316

TABLE C-1
The Prime Extended Character Set - Continued

Graphic	Mnemonic	Description	Binary	Decimal	Hex	Octal
O		Uppercase O	1100 1111	207	CF	317
P		Uppercase P	1101 0000	208	D0	320
Q		Uppercase Q	1101 0001	209	D1	321
R		Uppercase R	1101 0010	210	D2	322
S		Uppercase S	1101 0011	211	D3	323
T		Uppercase T	1101 0100	212	D4	324
U		Uppercase U	1101 0101	213	D5	325
V		Uppercase V	1101 0110	214	D6	326
W		Uppercase W	1101 0111	215	D7	327
X		Uppercase X	1101 1000	216	D8	330
Y		Uppercase Y	1101 1001	217	D9	331
Z		Uppercase Z	1101 1010	218	DA	332
[	LBKT	Left bracket	1101 1011	219	DB	333
\	REVS	Reverse slash, backslash	1101 1100	220	DC	334
]	RBKT	Right bracket	1101 1101	221	DD	335
^	CFLX	Circumflex	1101 1110	222	DE	336
_		Underline, underscore	1101 1111	223	DF	337
`	GRAV	Left single quote, grave accent	1110 0000	224	E0	340
a		Lowercase a	1110 0001	225	E1	341
b		Lowercase b	1110 0010	226	E2	342
c		Lowercase c	1110 0011	227	E3	343
d		Lowercase d	1110 0100	228	E4	344
e		Lowercase e	1110 0101	229	E5	345
f		Lowercase f	1110 0110	230	E6	346
g		Lowercase g	1110 0111	231	E7	347
h		Lowercase h	1110 1000	232	E8	350
i		Lowercase i	1110 1001	233	E9	351
j		Lowercase j	1110 1010	234	EA	352
k		Lowercase k	1110 1011	235	EB	353
l		Lowercase l	1110 1100	236	EC	354
m		Lowercase m	1110 1101	237	ED	355
n		Lowercase n	1110 1110	238	EE	356
o		Lowercase o	1110 1111	239	EF	357
p		Lowercase p	1111 0000	240	F0	360
q		Lowercase q	1111 0001	241	F1	361
r		Lowercase r	1111 0010	242	F2	362
s		Lowercase s	1111 0011	243	F3	363
t		Lowercase t	1111 0100	244	F4	364

TABLE C-1
The Prime Extended Character Set - Continued

Graphic	Mnemonic	Description	Binary	Decimal	Hex	Octal
u		Lowercase u	1111 0101	245	F5	365
v		Lowercase v	1111 0110	246	F6	366
w		Lowercase w	1111 0111	247	F7	367
x		Lowercase x	1111 1000	248	F8	370
y		Lowercase y	1111 1001	249	F9	371
z		Lowercase z	1111 1010	250	FA	372
{	LBCE	Left brace	1111 1011	251	FB	373
|	VERT	Vertical line	1111 1100	252	FC	374
}	RBCE	Right brace	1111 1101	253	FD	375
~	TIL	Tilde	1111 1110	254	FE	376
	DEL	Delete	1111 1111	255	FF	377

D

Error Messages

Error messages in this appendix are listed in two sections:

- Runtime error messages
- Batch error messages

Errors are listed alphabetically within each group, according to the first word that is constant. Leading asterisks and variable names are ignored in alphabetizing.

Runtime Error Messages

Descriptions of runtime error messages may be followed by labels enclosed either in parentheses or brackets. For example,

`Already exists.` **File System**
Attempt made to create an object with the same name as one already existing. (CREA$$) (SRCH$$) [E$EXST]

The label in parentheses is usually the name of a subroutine or group of subroutines; the label in brackets is the name of an error code. If a call to the subroutine is problematic, the error code is invoked. The error message is connected to the error code and is displayed at the user's terminal.

`ACCESS VIOLATION` **64V mode**
Attempt to perform operations in segments to which user has no right.

`ACL too big.` **File System**
Either ACL contains more than 32 entries, or the ACL exceeds space limitations. (AC$SET, AC$CHG) [E$ACBG]

`****AD` **R-mode function**
Overflow or underflow in double-precision addition/subtraction (A$66,S$66).

`All file units in use.` **File System**
User has requested use of a file-unit when he already has the maximum allowable number of file-units open, or the system has exhausted its pool of available units. (Search subroutines) [E$FUIU]

`ALL REMOTE UNITS IN USE` **File System**
Attempt made to assign a remote unit when none are available. (Network error) [E$FUIU]

`**** ALOG/ALOG 10 - ARGUMENT <=0` **V-mode function**
Argument not greater than zero used in logarithm (ALOG, ALOG 10) function.

`filename` `ALREADY EXISTS` **Old file call**
Attempt made to create a file or directory with the same name as one already existing.

`Already exists.` **File System**
Attempt made to create an object with the same name as one already existing. (CREA$$, SRCH$$) [E$EXST]

`****AT` **R-mode function**
Both arguments are zero in the ATAN2 function.

`**** ATAN2 - BOTH ARGUMENTS = 0` **V-mode function**
Both arguments are zero in the ATAN2 function.

`**** ATTDEV - BAD UNIT` **V-mode call**
Incorrect logical device unit number in the ATTDEV subroutine call.

`BAD CALL TO SEARCH` **Old file call**
Error in calling the SEARCH subroutine; for example, incorrect parameter.

`Bad command format` **PRIMOS**
User has issued an illegal command line. Command is ignored. [E$CMND]

`BAD DAM FILE` **Old file call**
The DAM file specified has been corrupted, either by the programmer or by a system problem. [SS]

`Bad DAM file.` **File System**
The DAM file specified has been corrupted, either by the programmer or by a system problem. (PRWF$$, SRCH$$) [E$BDAM]

`Bad dope vector.` **FORTRAN I/O**
Your program may have overwritten itself. If not, this message may indicate a compiler or library error. [E$BDV]

`Bad format.` **FORTRAN I/O**
Your program may have overwritten itself. If not, this message may indicate a compiler or library error. [E$FER]

`Bad key in call.` **PRIMOS**
Incorrect key value specified in subroutine argument by a user's program. [E$BKEY]

`Bad LUBTL entry.` **FORTRAN I/O**
Your program has possibly overwritten itself or part of the library. [E$BLUE]

`BAD PARAMETER` **Old file call**
Incorrect parameter value in subroutine call. [SA]

`Bad parameter.` **PRIMOS**
Incorrect parameter value in subroutine call. [E$BPAR]

`Bad password.` **File System**
Incorrect password specified in ATCH$$ subroutine. In pre-Rev. 19.0 systems, you return to PRIMOS level attached to no directory. In Rev. 19.0 and later systems, you return to PRIMOS level attached to the home directory. The home directory is either the directory from which you activated the ATCH$$ subroutine or another directory previously defined as home. (AT$ subroutines) [E$BPAS]

`Bad remote password.` **PRIMENET**
An attempt was made to log in to another system using an invalid password. [E$BRPA]

`BAD RTNREC` **PRIMOS**
System error.

`Bad segment directory unit.` **File System**
Error generated in accessing segment directory, that is, PRIMOS file-unit specified is not a segment directory. (SRCH$$) [E$BSUN]

`Bad stack format signaling.` **PRIMOS**
Condition mechanism cannot perform requested action because the command processor stack has been damaged (system error). Command environment is initialized and user returned to PRIMOS command level. [E$STKF, E$STKS]

`BAD SVC` **PRIMOS**
Bad supervisor call. In FORTRAN, usually caused by program writing over itself.

`Bad truncate of segment directory.` **File System**
Error encountered in truncating segment directory. (SGDR$$) [E$BTRAN]

`Bad unit number.` **File System**
PRIMOS file-unit number specified is invalid because it is outside legal range. (PRWF$$, RDEN$$, SRCH$$, SGDR$$). [E$BUNT]

`Bad use of EXIT.` **PRIMOS**
The condition mechanism sends this fatal message. You return to PRIMOS command level. [E$NEXP]

`Beginning of file.` **File System**
An attempt was made to access locations before the beginning of the file. (PRWF$$, RDEN$$, SGDR$$) [E$BOF]

`****BN` *n* **R-mode function**
Device error in REWIND command on FORTRAN logical unit *n*.

`Buffer too small.` **File System**

Buffer as defined is not large enough to accommodate entry to be read into it. (RDEN$$) [E$BFTS]

`Cannot access like reference.` **File System**

Object specified in the –LIKE option of SET_ACCESS could not be accessed for some reason. (AC$LIK) [E$LRNA]

`Category protects MFD.` **File System**

An attempt was made to delete an access category that protects the MFD. The MFD must be removed from the category before the category can be deleted. (CAT$DL) [E$CPMF]

`Command line more than 160 characters.` **PRIMOS**

A command line of more than 160 characters has been received. The command is not executed, and you return to PRIMOS command level. [E$TRCL]

`Command line truncated.` **PRIMOS**

A command line has exceeded the limit of 160 characters.

`Concealed stack overflow.` **PRIMOS**

System error. (Generally sent by the condition mechanism.) [E$CSOV]

`Crawlout unwind failed.` **PRIMOS**

System error. (Generally sent by the condition mechanism.) [E$CRUN]

`**** DATAN - BAD ARGUMENT` **V-mode function**

The second argument in the DATAN2 function is zero.

`****DE` **R-mode function**

The exponent of a double-precision number has overflowed.

`The device is in use.` **File System**

An attempt was made to ASSIGN a device currently assigned to another user. [E$DVIU]

`Device not assigned.` **File System**

An attempt was made to perform I/O operations on a device before assigning that device. [E$NASS]

`Device is not started.` **File System**

An attempt was made to access a disk not physically or logically connected to the system. If disk must be accessed, the system operator must start it up. [E$DNS]

`**** DEXP - ARGUMENT TOO LARGE` **V-mode function**

The argument of the DEXP function is too large; that is, it will give a result outside the legal range.

`**** DEXP - OVERFLOW*UNDERFLOW` **V-mode function**

An overflow or underflow condition occurred in calculating the DEXP function.

`The directory is damaged.` **File System**

The directory has become corrupted. (All file system subroutines.) [E$BUFD]

`The directory is not empty.` **File System**
An attempt was made to delete a non-empty directory. (SRCH$$, FIL$DL) [E$DNTE]

`Directory still contains access categories.` **File System**
You have tried to convert an ACL directory to a password directory, but the directory still contains one or more access categories; having these access categories is not allowed. (AC$RVT) [E$CATF]

`Directory still contains ACL subdirectories.` **File System**
You have tried to convert an ACL directory to a password directory, but the directory still contains one or more ACL subdirectories; having these subdirectories is not allowed. (AC$RVT) [E$ADRF]

`Directory too large.` **File System**
An attempt has been made to add too large an entry to a directory. (CREA$$, SRCH$$, AC$SET, AC$CHG) [E$FDFL]

`Disk format does not support this revision of PRIMOS.` **File System**
An attempt was made to convert a password directory to an ACL directory on a pre-Rev. 19 disk. (AC$SET, AC$DFT) [E$ST19]

`DISK FULL` **Old file call**
There is no more room for creating/extending any type of file on disk. [DJ]

`Disk has been shut down.` **File System**
The disk has been shut down. [E$SHDN]

`The disk is full.` **File System**
There is no more room for creating/extending any type of file on disk. (CREA$$, PRWF$$, SRCH$$, SGDR$$) [E$DKFL]

Note

Space may be made available. Use the PRIMOS commands ATTACH, LD, and DELETE to remove files that are no longer needed.

`Disk I/O error` **File System**
A read/write error was encountered in accessing disk. You return immediately to PRIMOS level. Not correctable by applications programmer. If this happens, notify your System Administrator. (All file system subroutines) [E$DISK]

`Disk is write-protected.` **File System**
An attempt has been made to write to a disk that is WRITE-protected. (All file system subroutines) [E$WTPR]

`DK ERROR` **Old file call**
A read/write error was encountered in accessing the disk. Try again; if you receive the same error, contact your System Administrator. [WB]

`****DL` **R-mode function**
Argument was not greater than zero in DLOG or DLOG2 function.

`**** DLOG*DLOG2 - ARGUMENT <=0` **V-mode function**
Argument not greater than zero was used in DLOG or DLOG2 function.

`****DN n` **R-mode function**
Device error (end of file) on FORTRAN logical unit *n*.

`**** DSIN*DCOS - ARGUMENT RANGE ERROR` **V-mode function**
Argument outside legal range for DSIN or DCOS function.

`**** DSQRT - ARGUMENT <0` **V-mode function**
Negative argument in DSQRT function.

`****DT` **R-mode function**
Second argument is zero in DATAN2 function. (D$22)

`DUPLICATE NAME` **Old file call**
Attempt to create/rename a file with the name of an existing file. [CZ]

`****DZ` **R-mode function**
Attempt to divide by zero (double-precision).

`End of file.` **File System**
Attempt to access location after the end of file. (PRWF$$, RDEN$$, SGDR$$) [E$EOF]

`****EQ` **R-mode function**
Exponent overflow. (A$81)

`Entry is an access category.` **File System**
An attempt was made to open an access category, as, for example, with the command SLIST. (SRCH$$) [E$IACL]

`****EX` **R-mode function**
Exponent function value too large in EXP or DEXP function.

`**** EXP - ARGUMENT TOO LARGE` **V-mode function**
The argument of the EXP function is too large, that is, it will give a result outside the legal range.

`**** EXP - OVERFLOW` **V-mode function**
Overflow occurred in calculating the EXP function.

`Fatal error in crawlout.` **PRIMOS**
System error. [E$CRWL]

`****FE` **R-mode function**
Error in FORMAT statement. FORMAT statements are not completely checked at compile time. (F$IO)

`File in use.` **File System**
The message may have two causes. First, you may have attempted to open a file already opened by you or by some other user. Second, you may have attempted to set a quota on a directory that currently has none and to which someone is attached or in which someone has a file open. (SRCH$$) [E$FDEL]

Note

At Rev. 18 and later, FUTIL no longer closes open file-units when it is invoked. Therefore, command files that depend on FUTIL to close units may receive `File in Use` or `File Open on Delete` messages. To avoid this message, close files explicitly, using the CLOSE command.

`File is delete-protected.` **File System**
The delete-protect switch has been set. (SRCH$$, FIL$DL) [E$DLPR]

`The file is too long.` **File System**
Attempt made to increase size of segment directory beyond size limit. (SGDR$$) [E$FITB]

`File open on delete` **File System**
Attempt made to delete a file that is open. (SRCH$$, FIL$DL) [E$FDEL]

Note

At Rev. 18 and later, FUTIL no longer closes open file-units when it is invoked. Therefore, command files which depend on FUTIL to close units may receive `File in Use` or `File Open on Delete` messages. To avoid this message, close files explicitly, using the CLOSE command.

`****FN` *n* **R-mode function**
Device error in BACKSPACE command on FORTRAN logical unit *n*.

`**** F$BN - BAD LOGICAL UNIT` **V-mode function**
FORTRAN logical unit number out of range.

`**** F$IO - FORMAT ERROR` **V-mode function**
Incorrect FORMAT statement. FORMAT statements are not completely checked at compile time.

`**** F$IO - FORMAT*DATA MISMATCH` **V-mode function**
Input data does not correspond to FORMAT statement.

`**** F$IO - NULL READ UNIT` **V-mode function**
FORTRAN logical unit for READ statement not configured properly.

`F$IOBF overflow.` **FORTRAN I/O**
You are trying to input or output more data than the internal buffer in the I/O subroutines can hold. A discussion of possible solutions appears in the *Subroutines Reference IV: Libraries and I/O.* [E$BKOV]

`Format/data mismatch.` **FORTRAN I/O**
A program contains a format/data mismatch; that is, format is numeric and data is alpha. If you are using formatted I/O, the data item you are inputting or outputting is not the type (such as integer, real, or character) that you specified in your format statement. If you are using list-directed input, any character data must be given in quotation marks. You may also have tried to read a real number into an integer variable. [E$FDMM]

`***II` **R-mode function**
Exponentiation exceeds integer size. (E$11)

`**** I**I - ARGUMENT ERROR` **V-mode function**
Exponentiation exceeds integer size.

`Illegal access mode.` **File System**
The access portion of an access pair contains an unknown access mnemonic. (AC$SET, AC$CHG) [E$BMOD]

`Illegal identifier.` **File System**
The identifier portion of an access pair contains an illegal user ID or group ID. (AC$SET, AC$CHG) [E$BID]

`ILLEGAL INSTRUCTION AT` *`octal-location`* **R mode**
An instruction at *octal-location* cannot be identified by the computer. This message appears only with R-mode programs compiled and linked on pre-Rev. 17 systems.

`Illegal name.` **File System**
Illegal name specified for a file or directory. (CREA$$, SRCH$$) [E$BNAM]

`Illegal remote reference.` **File System**
Attempt to perform network operations by user not on network or attempt to initiate a phantom from a remote partition. [E$IREM]

`Illegal treename.` **File System**
The string specified for a pathname is syntactically incorrect. [E$ITRE]

`****IM` **R-mode function**
Overflow or underflow occurred during a multiply. (M$11, E$11)

`filename IN USE.` **Old file call**
Attempt made to open a file already opened or to close/delete a file opened by another user, and so forth. [SI]

`Incorrect access control list format.` **File System**
ACL specified in SET_ACCESS or EDIT_ACCESS was not in proper format. This message usually results from an omitted colon between the identifier and the access rights. (AC$SET, AC$CHG) [E$BACL]

`Incorrect version number.` **File System**
A version number was passed to an ACL routine that was not recognized. (AC$SET, AC$CHG, AC$LST, GETID$) [E$BVER]

`Insufficient access rights.` **PRIMOS**
User does not have necessary access rights to file system object, or to perform the action desired. [E$NRIT]

`Invalid argument to command.` **PRIMOS**
A command has been issued with an illegal argument. The command is not executed. [E$BARG]

`Invalid segment number.` **File System**
Attempt made to access segment number outside valid range. [E$BSGN]

`****I/O error on logical unit` *`n`* **PRIMOS**
This FORTRAN error message is usually followed by a second message that gives more precise information on the problem. Two points to remember are

- FORTRAN's method of identifying logical units does not necessarily match the unit numbers given by the STATUS UNITS command.
- FORTRAN may not consider a file-unit open unless it is open in the needed mode. (For example, a file opened for reading only is still considered closed for writing.)

`I/O error or device interrupt.` **PRIMOS**
An external device has generated an interface line defined to cause an interrupt of the user program. [E$IEDI]

`****LG` **R-mode function**
Argument not greater than zero in ALOG or ALOG10 function.

`Like reference not found.` **File System**
A reference used with the –LIKE option of SET_ACCESS cannot be located. (AC$LIK) [E$LRNF]

`Max number of users exceeded.` **PRIMOS**
The maximum allowable number of users is already using the system. (This may mean that the operator has used the MAXUSR command to decrease the number of users temporarily.)

`Maximum quota exceeded.` **File System**
You have tried to store a number of records in a directory that exceeds the maximum number of records allowed for the directory. [E$MXQB]

`Maximum remote users exceeded.` **File System**
No more users may access the network. [E$TMRU]

`Name is too long.` **File System**
Length of name in argument list exceeds 32 characters. [E$NMLG]

`Network error detected.` **PRIMENET**
An error has occurred in PRIMENET while attempting to process a remote request. [E$NETE]

`No driver for device.` **FORTRAN I/O**
You are trying to use a device for which IOCS does not have a driver (that is, a subroutine that interfaces with a device). [E$NDFD]

`No information.` **File System**
You have tried to access a file system object and you could not. Some common reasons include insufficient access rights, non-existent object, and wrong type. This message does not reveal whether the specified object exists. [E$NINF]

`No phantoms are available.` **PRIMOS**
An attempt has been made to spawn a phantom, but the maximum allowable number of phantoms has been reached. (PHNTM$) [E$NPHA]

`No NPX slaves available.` **PRIMENET**
A remote reference to has been attempted but no slave processes are available on the remote system. [E$NSLA]

`No on-unit found.` **Condition mechanism**
Condition mechanism cannot take action. You return to PRIMOS command level. [E$NOON]

`No room.` **File System**
You have tried to add to a table of assignable devices with a DISKS or ASSIGN AMLC command, and the table is already filled. [E$ROOM]

`No room in output buffer.` **PRIMENET**
An error has occurred in PRIMENET while attempting to process a remote request. [E$NROB]

`No timer.` **File System**
Clock not started. System error. [E$NTIM]

`NO UFD ATTACHED` **Old file call**
You are not attached to a directory. Usually occurs after attempt to attach with a bad password in pre-Rev. 19.0 systems. [AL, SL]

`No UFD attached.` **File System**
You are not attached to a directory. Usually occurs after attempt to attach with a bad password in pre-Rev. 19.0 systems. (ATCH$$, CREA$$, GPAS$$, SATR$$, SRCH$$) [E$NATT]

`Not a file or a directory.` **File System**
You attempted to protect an access category with an ACL. Only files, directories, and segment directories can be protected by an ACL. (AC$SET, AC$CHG, AC$LIK, AC$CAT) [E$NTFD]

`Not a quota disk.` **File System**
You attempted to set or list a quota on a pre-Rev. 19 disk. (Q$SET) [E$NOQD]

`Not a segment directory.` **File System**
You attempted to perform segment directory operations on a file system object that is not a segment directory. (SRCH$$) [E$NTSD]

`NOT A UFD.` **Old file call**
You attempted to perform directory operations on a file that is not a directory. [AR]

`Not a UFD` **File System**
You attempted to perform directory operations on a file that is not a directory. (ATCH$$, GPAS$$, SRCH$$). [E$NTUD]

`Not an access category.` **File System**
This is not an access category. (AC$CAT) [E$NCAT]

`Not an ACL directory.` **File System**
You have attempted to use an ACL command (other than SET_ACCESS) on a password directory. [E$NACL]

`device-name NOT ASSIGNED` **PRIMOS**
User program has attempted to access an I/O device that has not been assigned to the user by a PRIMOS command.

`filename NOT FOUND` **Old file call**
File specified in subroutine call not found. [AH, SH]

`Not found. filename` **File System**
File specified in subroutine call not found. (Any file system subroutine.) [E$FNTF]

`Not found in segment directory.` **File System**
Filename specified in subroutine call not found in specified segment directory. (SRCH$$, SGDR$$, SRSFX$) [E$FNTS]

`NULL READ UNIT` **PRIMOS**
Program has attempted to read with a bad unit number. This may be caused by the program overwriting itself (array out of bounds).

`Object is category-protected.` **File System**
You attempted to use EDIT_ACCESS on an object currently protected by an access category. (AC$CHG) [E$CTPR]

`Object is default-protected.` **File System**
You attempted to use EDIT_ACCESS on an object that is currently default protected. (AC$CHG) [E$DFPR]

`Operation illegal on directory.` **File System**
User has tried to perform an illegal operation (such as editing) on a directory. [$DIRE]

`Operation partially blocked.` **PRIMENET**
An error has occurred in PRIMENET while attempting to process a remote request. [E$PRTL]

`Operation unsuccessful.` **PRIMENET**
An error has occurred in PRIMENET while attempting to process a remote request. [E$NSUC]

`****PA n` **R-mode function**
PAUSE statement *n* (octal) encountered during program execution.

`Parent not an ACL directory.` **File System**
An attempt was made to convert a password-protected directory to an ACL directory, but the parent of the directory being converted was a password directory. (AC$SET, AC$DFT) [E$PNAC]

`**** PAUSE n` **V-mode function**
PAUSE statement *n* (octal) encountered during program execution.

`PIO instruction did not skip.` **PRIMOS**
An invalid PIO sequence has been issued (for example, attempting to start DMX activity before loading the DMX channel address register). [E$DNSK]

`Pointer mismatch found.` **File System**
Internal file pointers have become corrupted. No user remedial action possible. System Administrator must correct. [E$PTRM]

`Priority ACL not found.` **File System**
No priority ACL exists for the partition specified in a LIST_PRIORITY_ACCESS command. (PA$LST) [E$PANF]

`Procedure not found.` **PRIMENET**
An attempt has been made to call a PRIMOS routine that does not exist on a remote system. This error can happen when there is a call from a new to an old Rev., and it is generated when a slave takes a linkage fault. [E$PNTF]

`Program halt at` *`segment no./word no.`* **R mode and 64V mode**
Program control has been lost. The program has probably overwritten itself or the load was incomplete (R mode).

`PRWFIL BOF` **Old file call**
Attempt by PRWFIL subroutine to access location before beginning of file. [PG]

`PRWFIL EOF` **Old file call**
Attempt by PRWFIL subroutine to access location after end of file. [PE]

`PRWFIL POINTER MISMATCH` **Old file call**
The internal file pointers in the PRWFIL subroutine have become corrupted.

`PRWFIL UNIT NOT OPEN` **Old file call**
The PRWFIL subroutine is attempting to perform operations by using a PRIMOS file-unit number on which no file is open.

`PTR MISMATCH` **File System**
Internal file pointers have become corrupted. No user remedial action possible. Consult your System Administrator. (ATCH$$, CREA$$, GPAS$$, PRWF$$, RDEN$$, SATR$$, SRCH$$, SGDR$$)

`Quota set below current usage.` **File System**
You have set a quota on a directory, but the directory already contains more records than the quota allows. This is a warning. (Q$SET) [E$QEXC]

`The remote line is down.` **PRIMENET**
The network connection between the local system and the remote system cannot be established. [E$RLDN]

`Remote system not up.` **PRIMENET**
An attempt has been made to access a remote system that is down. [E$RSNU]

`Requires receive enabled.` **PRIMOS**
An attempt has been made to send a message without receive being enabled. [E$NRCV]

`****RI` **R-mode function**
Argument is too large for real-to-integer conversion. (C$12)

`****RN` *n* **R-mode function**
Device error or the end of file in the READ statement on FORTRAN logical unit *n*.

`****SE` **R-mode function**
Single-precision exponent overflow.

`SEG-DIR ER` **Old file call**
Error encountered in segment directory operation. [SQ]

`Segment directory error.` **PRIMOS**
Error encountered in segment directory operation. [E$SDER]

`Segment directory unit not open.` **File System**
Attempt has been made to reference a segment directory that is not open. (SRCH$$) [E$SUNO]

`Semaphore overflow.` **File System**
System error. [E$SEMO]

`**** SIN/COS - ARGUMENT TOO LARGE` **V-mode function**
Argument too large for SIN or COS function.

`Slave validation error.` **PRIMENET**
Either you are trying to access a remote system that requires user validation, and you did not issue an ADD_REMOTE_ID command (to establish a remote ID), or a previous ADD_REMOTE_ID command established a user ID, project ID, or password that was invalid on the remote system. [E$SVAL]

`****SQ` **R-mode function**
Negative argument in SQRT or DSQRT function.

`**** SQRT - ARGUMENT<0` **V-mode function**
Negative argument in SQRT function.

`****ST` *n* **R-mode function**
STOP statement *n* (octal) encountered during program execution.

`Stack overflow in crawlout.` **PRIMOS**
System error. [E$CROV]

`**** STOP` *n* **V-mode function**
STOP statement *n* (octal) encountered during program execution.

`****SZ` **R-mode function**
Attempt to divide by zero (single-precision).

`System console command only.` **PRIMOS**
The command issued must be from the supervisor terminal. [E$SCCM]

`Top-level directory not found or inaccessible.` **File System**
An attempt to attach to a top-level directory has failed either because the directory does not exist, because PRIMOS cannot reach the system where it does exist (if a remote system is down), or

because you do not have the right (Use, or U, access) to attach to it. You remain attached to the previous current directory. (AT$ANY, AT$) [E$NFAS]

`Unable to find fault frame.` **Condition mechanism**
A call was made to CNSIG$, but CNSIG$ could not find that any condition had been raised.

`UNIT IN USE` **Old file call**
Attempt to open a file on a PRIMOS file-unit already in use. [SI]

`Unit in use.` **File System**
Attempt to open a file on a PRIMOS file-unit already in use. (SRCH$$) [E$UIUS]

`UNIT NOT OPEN` **Old file call**
Attempt to perform operations with a file-unit number on which no file has been opened or which is opened in the wrong mode (for example, a read to a unit open only for writing). [PD, SD]

`Unit not open.` **File System**
Attempt to perform operations with a file-unit number on which no file has been opened or which is opened in the wrong mode (for example, a read to a unit open only for writing). (PRWF$$, RDEN$$, SRCH$$, SGDR$$) [E$UNOP]

`UNIT OPEN ON DELETE` **Old file call**
Attempt to delete file without having first closed it. [SD]

`*** Unknown addressee.` **PRIMOS**
The user ID you have specified to receive a message is not the ID of a logged-in user. [E$UADR]

`*** User` *`usernumber`* `busy, please wait.` **PRIMOS**
The user to whom you have sent a message already has a deferred message waiting. Only one deferred message is allowed. You must send your message again. [E$UBSY]

`*** User` *`usernumber`* `not receiving now.` **PRIMOS**
The receive state of the user to whom you wish to send a message is either DEFER or REJECT. [E$UNRV]

`User unable to receive messages.` **PRIMOS**
An attempt to send a message to another user was not successful. [E$UDEF]

`Warm start occurred.` **PRIMOS**
Any device connected to GPPI has received a reset signal. [E$WMST]

`Wrong file type.` **FORTRAN I/O**
You are trying to use direct access I/O when you declared the file to be sequential in the OPEN statement, or vice versa. [E$WFT]

`****WN` *`n`* **R-mode function**
Device error or end of file in WRITE statement on FORTRAN logical unit *n*.

`****XX` **R-mode function**
Integer argument >32767.

Batch Error Messages

This list of Batch messages and their meanings does not include FIXBAT messages. The nature of each message is indicated in parentheses at the beginning of each explanation. The following types of messages exist:

Type	*Meaning*
Response	The message is displayed in response to a command or in addition to the normal response of a command. These are informative messages only; they do not indicate that anything is wrong with your command or the Batch subsystem.
Warning	The message is displayed to indicate that some part of your request or something about the current state of the Batch subsystem may affect the successful honoring of your request.
Fatal	The message is displayed to indicate that your request failed due either to an error on your part or to the temporary inability of the Batch subsystem to honor your request.
Severe	The message is displayed to indicate a severe error involving the Batch database. Typically, you have to run FIXBAT, INIT, or even FIX_DISK to repair the problem. (Running INIT causes all Batch job data to be lost.)
Message	The message is sent to the supervisor terminal. The severity of the message is also indicated.
Query	The message is displayed to elicit a response from you. Answer YES or NO, as appropriate.

The messages, listed in alphabetical order, start with those messages that begin with variable names.

`extra-text` `seen when end-of-line expected.`

(Fatal) *extra-text* has been entered when there should have been no more text (at the end of line). If this message occurs after you enter the BATGEN command, you return to PRIMOS, and PRIMOS displays the `ER!` prompt. If this message occurs in response to a BATGEN command or subcommand, an interactive user is left in command/subcommand mode, whereas a command file or CPL program is aborted.

`queue-priority` `is out of range. -PRIORITY`

(Fatal) The number *queue-priority*, supplied with the –PRIORITY option, was out of range. The range for *queue-priority* is 0 through 9. Resubmit the job, using a valid value for *queue-priority*.

`unit-number` `is out of range. -FUNIT`

(Fatal) The number *unit-number*, supplied with the –FUNIT option, was out of range. The range for *unit-number* is 1 through 128. Resubmit the job, using a valid value for *unit-number*.

`Another user is running FIXBAT or INIT.`

(Message, Fatal) The Batch monitor cannot start up because another user is running the FIXBAT program or the INIT program. If you know who is running the program, wait until the program finishes running, then restart the Batch monitor using BATCH –START. If you do not know who is running the program, periodically attempt to restart the Batch monitor.

`Bad $$ command.`

(Fatal) A file submitted using the JOB command has a $$ line as the first noncomment line, but the $$ command is not a $$ JOB command. Change the file so that the $$ line is valid. The use of $$ is reserved for future expansion by BATCH.

`Bad queue control file.`

(Severe) One of the Batch subsystem database files is inaccessible or has a bad format. The Batch subsystem is inoperative until the database is fixed.

`Bad queue definition file.`

(Fatal) A file referenced by BATGEN does not comply to format requirements; it is not a valid queue definition file. If this error occurs anywhere other than the BATGEN program, then the system Batch definition file has been overwritten with invalid data, and the Batch subsystem is inoperative.

`*BATCH* Database invalid.`

(Message, Severe) The monitor logs itself out after sending this message, and the Batch system is left inoperative. (Users receive error messages if they try to invoke JOB or BATCH.) The System Administrator should determine what the error is and fix it if possible. If the Batch monitor generates a command output (log) file, that should reveal the source of the error. The file is named O_LOG in BATCHQ (if the file BATCHQ>START_BATCH_MONITOR.COMI runs FIXBAT.SAVE with a –STARTUP argument other than NOLOG).

In general, if the exact cause of the problem is not known (such as a Pointer mismatch error in the database or a disk write-protected error), run FIXBAT. If that fails, resume the BATCHQ>INIT program using the –RESET_QUEUES option to reinitialize the entire database. If this doesn't work, there are probably disk errors. If it does work, redefine the Batch queues using BATGEN and start the Batch monitor up again. (All job data is deleted by the INIT program.)

`BATDEF file is missing.`

(Message, Fatal) The queue definition file, which is the crux of the database, is not present. The monitor logs itself out after sending this message. The System Administrator should use BATGEN to generate a new BATDEF file.

`Can't log error.`

(Message, Severe) An error has occurred that the monitor cannot record. (This message generally accompanies other severe error messages.)

`Can't start batch job!`

(Message, Fatal) The Batch monitor has not been started from the supervisor terminal, and it cannot log in processes under different login names or log out other processes. The monitor logs itself out after sending this message. Issue the BATCH –START command from the supervisor terminal if this situation occurs.

`(Changes made)`

(Response) The changes specified in a JOB –CHANGE operation have been made. If the job is initiated after the changes are made, then it executes with the specified changes in place. The job status is displayed after the `Changes made` message is displayed.

`Command or CPL file required as first arg. on submission.`

(Fatal) A JOB command has been given with job options (such as –HOME, –PRIORITY, –CPTIME, and so on) but no CPL program name or command file name was specified. The command format is

JOB ***pathname*** **[*options*]**

`Cpu limit must be specified.`

(Fatal) The required –CPTIME option has not been supplied for a queue specified by a –QUEUE option during job submission. (That is, the default CPU limit for that queue is greater than the maximum CPU limit for that queue.) Resubmit the job with the –CPTIME option specified. To determine the maximum limits for queues, use BATGEN –DISPLAY.

`Creating new batch definition file:` *`pathname`* `(BATGEN)`

(Response) The *pathname* specified does not exist. When the FILE command is given, it creates the specified file and puts the Batch queue definitions in it. BATGEN initializes its environment when it cannot find *pathname*; therefore no queues are defined.

`Date and time not set. (Batch)`

(Fatal) Either BATGEN or JOB command or a RESUME BATCHQ>INIT command has been issued from the supervisor terminal before the system date and time were set. These parts of the Batch system cannot be run until the system date and time are set, using the SETIME command from the supervisor terminal.

`Elapsed time limit must be specified.`

(Fatal) The –ETIME option has not been specified for a particular queue. (That is, the default elapsed time limit for that queue is greater than the maximum elapsed time limit for that queue.) Resubmit the job with the –ETIME option specified. To determine the maximum limits for queues, use BATGEN –DISPLAY.

`End of line.`

(Fatal) A required keyword or option was not present on the command line. The message generally contains more information on what is expected. Reenter the command with the additional requested information.

`End of line. Illegal` *`option-name`* `argument`

(Fatal) A job parameter option, *option-name*, was specified last on the JOB command line or on the $$ JOB line, but had no argument (end of line). Supply the information required by *option* when you reenter the command or modify the $$ JOB line accordingly.

`End of line. Queue name required`

(Fatal) A command entered while in BATGEN command mode requires a queue name. (ADD, MODIFY, BLOCK, UNBLOCK, and DELETE all require queue names.) Reenter the command with the queue name desired.

`End of line. Value required`

(Fatal) While in BATGEN subcommand mode, a subcommand that requires at least one numeric parameter has been issued, but no number was given. Subcommands requiring at least one numeric parameter are CPTIME, ETIME, FUNIT, PRIORITY, TIMESLICE, and RLEVEL. Note that the CPTIME and ETIME subcommands accept two parameters, both of which can be the keyword NONE, indicating no limits. Reenter the subcommand with the value desired. For example, TIMESLICE 10.

```
Enter queue characteristics:
$
```

(Response) The ADD or MODIFY command, given while in BATGEN command mode, has succeeded. You are now in BATGEN subcommand mode, identified by the `$` prompt instead of the > prompt used in BATGEN command mode. To reenter command mode from subcommand mode, use QUIT or RETURN. RETURN saves the information modified while in subcommand mode; QUIT discards it, asking for verification if any of it was changed.

`Environment modified, ok to quit?`

(Query) A QUIT command has been issued while in BATGEN command mode, after the environment was modified. Valid responses to this question are YES, NO, and OK. If YES or OK is the response, a subsequent START command reenters BATGEN command mode with no loss of information about the environment.

`Error:` *`message-text (program)`* `err=`*`nnnn`*

(Message, Severe) An error occurred in the Batch subsystem, encountered either by the Batch monitor, a Batch job, or a Batch user. Typically, other fatal error messages are sent to the supervisor terminal and the database is invalidated. Use FIXBAT to fix the database. If this fails, try running the INIT program. *nnnn* is the error code, *program* is the program or subroutine in Batch that encountered the error, and *message-text* is additional information on the error.

`Extraneous text on command line. (MONITOR)`

(Fatal) A bad command line exists in BATCHQ>START_BATCH_MONITOR.COMI. The command line should read RESUME MONITOR or RESUME MONITOR –HUSH, but some excess information currently follows the –HUSH option. Fix the command line in the file and restart the Batch monitor.

`File has no non-comment lines.` *`pathname`* `(JOB)`

(Fatal) A user has submitted a command file or CPL program, named *pathname* that either is empty or is made up entirely of comment lines.

`Force logout by operator.`

(Message, Response) The Batch monitor has been forcibly logged out. It sends this message to indicate that it has successfully logged out without leaving the Batch database in an indeterminate state.

`Home ufd required.`

(Fatal) The –HOME option was not present on the JOB command line or on the optional $$ JOB line during submission, and JOB was unable to determine the attach point of the submitting job. Resubmit the job, and include the –HOME option followed by the absolute pathname indicating where the job is to execute. If the pathname is too long to fit, use a shorter version of it when you resubmit the file. First, edit the file to include an ATTACH command with a relative pathname that attaches down through the remaining subdirectories to reach the destination. Then, resubmit the job, using the shortened version of the pathname.

`Home=`*`pathname`*

(Response) During job submission, the –HOME option has not been specified on the command line or in the file ($$ JOB), but the job was successfully submitted. The JOB command determined the home attach point of the submitting user to be *pathname*, and used this as the home attach point of the submitted job.

Note

JOB does not attempt to determine whether the user can attach to the home directory as owner. If the user cannot attach because of a bad password error or an insufficient access rights error, the job terminates, and a requested command output file is not produced.

`Illegal -CHANGE option.`

(Fatal) The options –QUEUE and –PRIORITY are invalid during a –CHANGE operation using the JOB command, because queue and queue priority of a job cannot be changed. Cancel or abort the job and resubmit it to the appropriate queue with the desired queue priority.

`Illegal answer.`
(Warning) This warning is displayed when the answer to a question is not YES, NO, or OK. The question is asked again. These questions are asked when you try to QUIT out of BATGEN command or subcommand mode after modifying the environment or queue.

`Illegal combination.` *`option`*
(Fatal) A job parameter (such as –ACCT, –HOME, –QUEUE, and so on) has been specified on the same JOB command line as an option to perform a certain command such as –CANCEL, –DISPLAY, –ABORT, and so on. *option* is the second (conflicting) option. Use separate JOB commands to perform separate functions.

`Illegal combination. -FUNIT (JOB)`
(Fatal) A CPL job has been submitted using the –FUNIT option. This option is not valid for CPL jobs. Resubmit the job without the –FUNIT option.

`Illegal limit.`
(Fatal) A parameter supplied to the –CPTIME or –ETIME option during job submission/changing is not a valid limit. That is, it is less than or equal to zero or is not a valid decimal number, and it is not the keyword NONE. Reenter the command with valid limits.

`Illegal name.`
(Fatal) One of the Batch programs is expecting a name or command, but instead it reads an unquoted token beginning with a hyphen (-), indicating that an option is present.

`Illegal number.` *`n`* `(BATGEN)`
(Warning) The numeric parameter *n* supplied for a BATGEN subcommand is not a valid decimal number. Reenter the line with a valid decimal number. (All numbers input by the Batch subsystem are decimal.) Subcommands that can return this error are CPTIME, ETIME, FUNIT, PRIORITY, TIMESLICE, and RLEVEL. Note that the CPTIME and ETIME subcommands accept the keyword NONE indicating no limits, but flag the number 0 as an invalid number. Also, these two subcommands interpret the numbers as numbers ranging 1 through 999999999, whereas the numbers for the other subcommands range 0 through 32767.

`Illegal number.` *`n`* `(JOB)`
(Fatal) The numeric argument *n* supplied for the –FUNIT or –PRIORITY option during job submission using the JOB command is not a valid decimal number. Reenter the command line with valid numeric parameters.

`Illegal option.`
(Fatal) One of the Batch programs is expecting an option, namely, an unquoted token beginning with a hyphen (-). Reenter the command line with a valid format.

`Illegal queue name.` *`queue-name`* `(BATGEN)`
(Warning) An attempt has been made to add a queue with a name (*queue-name*) that does not comply with filename rules. (These rules are that the first character must not be a digit; and the character set is limited to alphanumeric, and selected special characters). Reenter the command with a valid queue name. Note that a queue name of ALL is invalid, since the DELETE ALL command would otherwise be ambiguous.

`Illegal queue name.` *`queue-name`* `(JOB)`
(Fatal) The queue name (*queue-name*) specified after a –QUEUE option while submitting or changing a job does not comply with queue name format rules. Use BATGEN –STATUS or –DISPLAY to determine the names of valid queues.

`Illegal value. `*`value`*` (BATGEN)`
(Warning) The parameter *value* supplied for the BATGEN RLEVEL subcommand is not valid.

`In `*`pathname:`*
(Fatal) This opening phrase precedes JOB error messages when the errors originate in a $$ JOB line within the file *pathname*. The error message also includes the $$ JOB line itself.

`In the submission file:`
(Fatal) This opening phrase precedes JOB error messages when the errors originate in the $$ JOB line of a file, and the submission program cannot determine the file's pathname to display it.

`Incorrect username.`
(Fatal) The username of the submitting user does not match the username in the $$ JOB line of a file submitted using the JOB command. Edit the file and change the username in the $$ JOB line to the username of the submitter. Note that a username of * means that any user may submit the file.

`Info in "BATCHQ>ERROR.". (BILD$B)`
(Severe) The source of an error has been successfully written to the file BATCHQ>ERROR. for perusal by the System Administrator. (Note that the period is included in the pathname.) This message is usually preceded and followed by other severe error messages.

`*** Invalid batch database, please contact your system administrator.`
(Severe) This message means that the Batch subsystem program being run has detected an error (such as disk failure, pointer mismatch, or misprotected file) in the Batch system database. It flags the database as invalid. Notify the System Administrator, who has the responsibility for reinitializing the database, running FIXBAT, or running FIX_DISK, as appropriate. The BATCH and JOB commands are inoperative until the situation is resolved.

`Invalid -COMO pathname. `*`invalid pathname`*` (JOB)`
(Fatal) The format of the pathname specified with the –COMOUTPUT option to the JOB command is invalid.

`Invalid -DEFER option time. `*`invalid time`*` (JOB)`
(Fatal) The format of the time or date specified with the –DEFER option to the JOB command is invalid.

`Invalid project id. `*`invalid project id`*` (JOB)`
(Fatal) Either the format of the project ID specified with the –PROJECT option to the JOB command is invalid, or the user does not belong to the specified project.

`?Job `*`jobname (jobid) job-status.`*
(Warning) An attempt has been made to use the JOB command on a job named *jobname* with an internal job ID of *jobid*, but its status (*job-status*) prevented such an operation. Examples of such attempts include trying to restart a completed job and attempting to release a job that is not held.

`Job `*`jobname for username (jobid) job-status.`*
(Message, Response) The Batch monitor has changed the status of a job. (This message is not displayed when the monitor changes a restarted job back to waiting). *jobname* is the external name of the job, *username* is the submitting user, *jobid* is the internal job ID, and *job-status* is either aborted or completed.

`Job name required.`
(Fatal) A required job identifier (an internal job ID or external name) has not been specified with one of the following options: –CHANGE, –CANCEL, –ABORT, –RESTART, –HOLD, and –RELEASE. Reenter the command with the job identifier. For example,

```
JOB TOP -CHANGE -PRIORITY 9
JOB #10032 -ABORT
```

`(Job no longer restartable)`
(Response) A JOB –CANCEL has been performed on an executing job. The job itself was not canceled, but it has been flagged as being unrestartable. In this state, use of the –RESTART option aborts the job but does not restart it.

`(Job not changed.) Queue not found.` *`queue-name`* `(JOB)`
(Fatal) A request to change the characteristics of a job cannot be honored because the queue to which the job was submitted (queue *queue-name*) cannot be found in BATCHQ>BATDEF. This is an unusual error, but it can result if a queue is deleted at a particular moment during the JOB –CHANGE operation.

`Job not found.`
(Fatal) The job referred to in a JOB command such as –CHANGE, –CANCEL, –ABORT, –RESTART, –HOLD, or –RELEASE cannot be found by searching the active jobs list. This can mean one of three things: that no job exists with that name; that all jobs with that name have completed, aborted, or canceled; or that a job exists with that external name but the user making the request is not the same user who originally submitted the job.

`(Job not restartable)`
(Warning) A JOB –RESTART has been performed on an unrestartable job. An attempt is made to abort the job after this message is displayed.

`(Job restarted)`
(Response) A JOB –RESTART has been performed on a restartable job. Although an error message can appear after this message, the job is generally restarted unless a JOB –CANCEL or JOB –CHANGE –RESTART NO command is issued. One possible error message, `Insufficient access rights`, may appear if the user is logged in as SYSTEM or BATCH_SERVICE and has restarted another user's job from a user terminal, or if the process has recently logged out. `Not found` can also be displayed if the process is logged out.

`Job will be restarted.`
(Message, Response) This message is displayed when Batch is first started. The message is sent to the supervisor terminal after a `Job` *`jobname`* `for` *`username (jobid)`* `aborted/completed` message is sent. It means that the job is eligible for restarting, and is therefore being reset to the waiting state. The message generally indicates that the job is restarted following a system shutdown.

`*** Jobs are not being processed at this time.`
(Severe) If followed by `*** Please contact your system administrator immediately`, this message indicates that the Batch database has not been initialized or that something has happened (such as a disk head crash). If followed by `*** Please try again later`, it indicates that the Batch monitor was logged out using a method other than BATCH –STOP, but the Batch monitor verifies the validity of the database when it is restarted. In either case, the user is immediately returned to command mode, and the operation the user attempted is *not* performed. This message may be displayed by the BATCH command or by the JOB command.

`Monitor already started.`
(Response) Informs the operator that the monitor is already started.

`Monitor continued.`
(Response and Message both) The Batch monitor has been continued. Jobs may now be started up. This message is displayed as the result of a BATCH –CONTINUE command following an earlier BATCH –PAUSE command.

`Monitor in operation.`
(Message, Response) Tells the operator that the Batch monitor has finished fixing the database (by running FIXBAT) and is ready to process jobs.

`Monitor paused.`
(Response, Message) The Batch monitor has been paused or has been started up in the paused state. No jobs will be started up. Use BATCH –CONTINUE to continue the monitor.

`Monitor started up.`
(Response) Tells the operator that the monitor has been started up and is now going through an initialization phase.

`Multiple jobs with this name (use internal name).`
(Fatal) The job name used in the JOB command belongs to at least two jobs submitted by this user. The job ID (internal name) must be used in this case. Use JOB –STATUS to determine the internal and external names of all active jobs belonging to the user issuing the command.

`Multiple occurrence.`
(Fatal) An option has been specified twice either on the JOB command line or on the $$ JOB line during job submission or job changing (for example, JOB TEST –HOME HERE –HOME THERE). (If an option is specified once on the JOB line and once on the $$ JOB line, no error results; the parameter on the JOB line takes precedence.) Reenter the command, but specify each option only once.

`Multiple monitors illegal.`
(Message, Fatal) An attempt has been made to start up a second Batch monitor. The monitor sending this message logs out.

`Must be first option.`
(Fatal) The option –CHANGE, –CANCEL, –ABORT, –RESTART, –STATUS, –DISPLAY, –HOLD, or –RELEASE must be the first option on the JOB command line (after a sometimes-optional job identifier).

`My disk is full. Please help me.`
(Message, Warning) The Batch monitor has encountered a Disk Full condition while trying to initiate a job. It retries the job initiation every five minutes, sending this message after each unsuccessful attempt. This message causes a bell to ring at the supervisor terminal. Delete some files from the disk to free up space.

`My quota is exceeded. Please help me.`
(Message, Warning) The Batch monitor has encountered a Quota Full condition while trying to initiate a job. It retries the initiation every five minutes, sending this message after each unsuccessful attempt. This message causes a bell to ring at the supervisor terminal. Delete some files from the disk to free up space or raise the quota set on the BATCHQ directory.

`No active jobs`

(Response) Displayed by a JOB –DISPLAY or –STATUS command, this message indicates that there are no waiting, held, or executing jobs belonging to the user. If the user is SYSTEM or BATCH_SERVICE, then there are no jobs that are waiting, held, or executing in the entire system. The JOB command may also display `named` ***jobname***, with either `for user` ***username*** or `in system` appended to the message, depending on whether the user specified a job name and whether the user is privileged (logged-in as SYSTEM or BATCH_SERVICE).

`No configured queues.`

(Response) A BATGEN –STATUS or –DISPLAY command has been issued for a Batch definition file that has no defined queues.

`No job changes specified.`

(Fatal) The –CHANGE option has been given to the JOB command, but no actual changes were specified on the command line. Reenter the command, specifying changes to be made following the –CHANGE option on the command line.

`No longer executing.`

(Fatal) A JOB –ABORT or JOB –RESTART has been performed on a job that had execution status, but by the time the execution file was read in to determine the user number of the process, the process had disappeared. If the message `(Job restarted)` is displayed, then the job has restarted. Although the operation itself was unsuccessful, the desired results have been achieved.

`No queue available for job.`

(Fatal) A job submitted with the JOB command did not specify a –QUEUE option, and no suitable queue can be found. Suitability requirements include CPU and elapsed time limits within the confines of the queue, queue unblocked, and so on. Use the BATGEN –STATUS or –DISPLAY command to display a list of valid queues and their status.

`No queues have waiting or held jobs.`

(Response) A BATCH –DISPLAY command has been issued, and no queues had any waiting or held jobs. A queue can have one executing job not considered to be waiting or held.

`No right. Must be Batch Administrator or Console`

(Fatal) A –HOLD or –RELEASE operation has been attempted using the JOB command, and the user was not a Batch Administrator or using the supervisor terminal.

`No running jobs.`

(Response) A BATCH –DISPLAY command has been issued, and no jobs were running. Jobs can be waiting when no jobs are running, even when the monitor is running and there are free phantoms. There is always a small amount of turnaround time between submittal and execution of a job.

`Not an absolute treename.`

(Fatal) The home directory specified with the –HOME option of the JOB command is a relative pathname (beginning with *>). Resubmit the job, giving an absolute pathname after the –HOME option.

`Not your job.`

(Fatal) A job has been referenced using an internal name in the JOB command, but the job does not belong to the user making the reference. Use the JOB –STATUS command to obtain a list of all active jobs submitted by the requesting user.

`Note: the batch monitor is currently not starting up jobs.`
(Response) A Batch subsystem command has been issued while the Batch monitor is in a paused state. No jobs are started up while the monitor is paused.

`Null home ufd.`
(Fatal) The home directory specified with the –HOME option of the JOB command is a null string. Resubmit the job with an absolute pathname after the –HOME option.

`Operator stop.`
(Message, Response) The monitor received a stop request via a BATCH –STOP command. The monitor logs out after sending this message.

`Out of range.`
(Warning) A BATGEN subcommand has been given a numeric parameter that is out of range for that subcommand. The ranges are 1 through 128 for FUNIT, 0 through 9 for PRIORITY, 1 through 99 for TIMESLICE, and 0 through 7 for RLEVEL. Reenter the subcommand with the correct parameter.

`Please FILE.`
(Warning) A QUIT command has been issued while in BATGEN command mode, after the environment has been modified; the question `Environment modified, ok to quit?` has been asked, and the answer was NO. This message is a reminder to use FILE to create a modified environment.

`Please RETURN.`
(Warning) A QUIT subcommand has been given while in BATGEN subcommand mode, after the queue characteristics have been modified; the question `Queue definition modified, ok to quit?` has been asked, and the response was NO. This message is a reminder that the proper way to leave a subcommand session is to use the RETURN subcommand.

`Please wait.`
(Response) This message asks that the users be patient because the current program has been monopolizing the Batch database and other processes now have access to it. It is not a fatal error. It generally occurs only when a system is heavily loaded or when the user's program has a very low priority and does not run frequently.

`Queue` *`queue-name`* `already exists (`*`queue-status`*`).`
(Warning) While in BATGEN command mode, an attempt has been made to add a queue named *queue-name* that already existed. *queue-status* is blocked, unblocked, or flagged for deletion. To change the queue definition, use the MODIFY subcommand. However, if the queue is flagged for deletion, any attempt to block, unblock, modify, or display it causes the `Unknown queue name` message to be displayed.

`Queue` *`queue-name`* `deleted.`
(Message, Response) Queue *queue-name*, flagged for deletion in the BATDEF file, has just been deleted by the Batch monitor because the queue became empty.

`Queue` *`queue-name`* `flagged for deletion.`
(Warning) While in BATGEN command mode, an attempt has been made to delete a queue named *queue-name* that was already flagged for deletion. To allow the queue to disappear, use FILE to write out the BATDEF file. The queue disappears when it contains no more waiting, held, or executing jobs. It can then be added again.

`Queue blocked.`

(Fatal) The queue referred to by a –QUEUE option during job submission is currently blocked to new submissions. Try it again later, or use another queue. Use the BATGEN –STATUS to display a list of all queues.

`Queue definition modified, ok to quit?`

(Query) A QUIT subcommand has been issued while in BATGEN subcommand mode, and the characteristics of the queue being added or modified have been changed. Valid answers to this question are YES (or carriage return), NO, and OK.

`Queue deleted.`

(Fatal) A job has been submitted to an available queue, but by the time the file was copied and some other activities had taken place, the queue had been deleted. The job should be resubmitted to a different queue.

`Queue does not exist.`

(Fatal) The –QUEUE option on the JOB command line or the (optional) $$ JOB line referred to a queue that either did not exist or was in the process of being deleted (flagged for deletion). The BATGEN –STATUS command provides a list of currently available queues and the status of each queue.

`Queue files not corect rev.`

(Severe) The queue control files in the Batch database are not up to date with the revision of PRIMOS running on your system. Shut the Batch subsystem down, run the INIT program, and then start the Batch subsystem up again.

`Queue full.`

(Fatal) The queue to which the user has tried to submit the job already contains 10,000 jobs (whether active or inactive). The queue becomes available as soon as one of the jobs runs to completion. Meanwhile, use any other available queues.

`Register setting.`

(Fatal) Register settings are invalid in the Batch subsystem, except as part of a submitted file. Reenter the command line without the register setting. (A register setting is two octal numbers separated by a slash, such as 1/15.)

`Removed` *`queue-name`* `from BATDEF`

(Message, Response) This message is sent to the supervisor terminal when the Batch monitor finds, in the BATDEF file, a queue that is flagged for deletion but that has never been used. The message indicates that queue *queue-name* has been deleted from BATDEF and that no job data has been lost as a result.

`Specified value is out of range.`

(Fatal) The –CPTIME or –ETIME option specified during job submission or a –CHANGE operation is greater than the maximum allowed by the queue to which the job was submitted. This message is preceded by a message indicating the maximum limit for that queue (`Cpu limit is` *n* or `Elapsed time limit is` *n*). If the limits cannot be lowered so that the job can be successfully run, try a queue with higher limits.

`Stop request issued.`

(Response) The BATCH –STOP command has requested that the Batch monitor stop. Within 20 seconds the monitor sends an `Operator Stop.` message to the supervisor terminal and logs out.

`Syntax error. Register settings are illegal`
(Warning) A register setting (such as 1/15) has been found when no more information was expected. Reenter the command without register settings.

`This job cannot be restarted.`
(Response) This message is displayed by a JOB –DISPLAY command if the job being displayed cannot be restarted. A job is not restartable if a JOB –CANCEL command is issued for that job while it is executing, or if it is submitted with the –RESTART NO option. An attempt to restart the job aborts it without restarting it.

`(This job has already executed >`*n*`> time(s).)`
(Response) This message is displayed by a JOB –DISPLAY command if the job being displayed is executing and has already been executed (*n* times). The condition is caused by a JOB –RESTART or by a system cold start after shutdown while the job is executing.

`This job will be restarted.`
(Response) This message is displayed in response to a JOB –DISPLAY command if a JOB –RESTART has been done but the job is still executing. When the monitor sees that the job has aborted or completed, it returns the job to the waiting state.

`Too many options.`
(Fatal) At least two conflicting options are entered, such as JOB –DISPLAY –CHANGE or JOB TEST –ABORT –CANCEL. Use separate JOB commands to perform separate operations.

`Too many queues.`
(Warning) An attempt has been made, using the ADD command in BATGEN, to add a queue when there are already 16 defined queues (blocked, unblocked, *or* flagged for deletion).

`Unable to validate project.` *`project id`* `(JOB) Please contact system administrator. (JOB)`
(Fatal) A problem has arisen accessing the SAD.

`Unknown command.`
(Warning) An unrecognized command has been entered while in BATGEN command mode. The user is left in BATGEN command mode.

`Unknown option. -`*`option`*` entry (JOB)`

Usage:

```
JOB treename [-batch_options] [-ARGS cpl_args]
                    -HOme homedirectory
                    -QUEue queuename
                    -CPTime cpulimit
                    -ETime elapsedlimit
                    -DeFer date.time/-No-DeFer
                    -COMOutput comopathname/-No_COMOutput
                    -NotiFY/-No_NotiFY
                    -PROJect projid
             jobid -CHanGe [-batch_options] [-ARGS cpl_args]
                   -CANcel
                   -ABort
                   -ReSTart
                   -STatus
                   -DisPlay
```

(Fatal) User has requested unknown option.

`Unknown queue name.`

(Warning) A command entered while in BATGEN command mode refers to a queue that either does not exist or is flagged for deletion by the DELETE command.

`Unknown -STARTUP argument. `*`keyword`*` (FIXBAT)`

(Message, Fatal) The *keyword* supplied to the –STARTUP option is not SAVE, DELETE, SPOOL, or NOLOG.

`Unknown subcommand.`

(Warning) While in BATGEN subcommand mode, an unrecognized subcommand has been given. The user is left in subcommand mode.

`Unrecognized command line option.`

(Message, Fatal) The Batch monitor has been invoked with an unrecognized option on the command line. The only valid MONITOR option is –HUSH. Fix the file START_BATCH_MONITOR.COMI in the BATCHQ directory accordingly, and restart the Batch monitor using BATCH –START.

`Unrecognized option.`

(Fatal) BATGEN has been invoked with an unrecognized option on the command line. The only valid BATGEN options are –STATUS and –DISPLAY.

`Warning: -DEFER option time already passed. Job is now waiting (not deferred). (JOB)`

(Warning) User has submitted job with a defer time that is earlier than the current time.

`Warning: jobs are not being processed at this time.`

(Response) This message means that the Batch monitor is not running; therefore, any submitted jobs are not executed until it is started up.

Ed Command Summary

This appendix contains an alphabetic list of all ED commands and their functions. Acceptable command abbreviations are shown in red. For a detailed description of all commands, see the Editor Reference Section of the *New User's Guide To EDITOR and RUNOFF.*

Note

The *string* parameter in a command is any series of ASCII characters including leading, trailing, or embedded blanks. A semicolon terminates the command unless it appears within delimiters (as in the CHANGE, MODIFY, or GMODIFY commands) or is preceded by the escape character (^).

Command	*Function*
APPEND ***string***	Appends *string* to the end of the current line.
BOTTOM	Moves the pointer to a null line beyond the last line of the file.
BRIEF	Speeds editing by suppressing the (default) verification responses to certain ED commands.
CHANGE/*string-1*/*string-2*/[G] [*n*]	Replaces *string-1* with *string-2* for *n* lines. If G is omitted, only the first occurrence of *string-1* on each line is changed; if G is present, all occurrences on *n* lines are changed.
DELETE [*n*]	Deletes *n* lines, including the current line (default *n* = 1).
DELETE TO *string*	Deletes all lines up to but not including the line containing *string*.
DUNLOAD *filename* [*n*]	Deletes *n* lines from the current file and writes them into *filename*. (Default *n* = 1.)
DUNLOAD *filename* TO *string*	Same as DELETE...TO, but writes deleted lines into *filename*.
ERASE *character*	Sets erase character to *character*.
FILE [*filename*]	Writes the contents of the current file into *filename* and QUITs to PRIMOS. If *filename* is omitted, ED writes into the current file and displays its name.

FIND string	Moves the pointer down to the first line beginning with *string*.
FIND(*n*) ***string***	Moves the pointer down to the first line in which *string* starts in column *n*.
FNAME	Displays the name of the current file during an editing session.
GMODIFY	Allows the user to enter a string of subcommands that modify characters within a line.
IB string	The INSERT BEFORE command inserts *string* as a new line immediately before the current line.
INPUT $\left\{\begin{array}{c}\textbf{ASR}\\ \textbf{PTR}\\ \textbf{TTY}\end{array}\right\}$	Reads text from the specified input device: ASR (teletype paper-tape reader), PTR (high-speed paper tape reader) or TTY (terminal). Default is TTY.
INSERT ***string***	Inserts *string* after the current line.
KILL ***character***	Sets kill character to *character*.
LINESZ [*n*]	Changes maximum line size (that is, length) of both command lines and input lines. This length, which is 1024 at start-up, may be set in the range 10 through 1024.
LOAD ***filename***	Loads *filename* into the text following the current line.
LOCATE ***string***	Moves pointer forward to the first line containing *string*, which may contain leading and trailing blanks.
LOCATE ***string*, ***	Moves pointer forward to each occurrence of *string* between pointer's current position and the end of file.
MODE CKPAR	Displays characters as real characters if parity is on, as octal numbers (^*nnn*) if parity is off.
MODE COLUMN	Displays column numbers whenever INPUT mode is entered.
MODE COUNT [*start*] [*increment*] [*width*] $\left\{\begin{array}{l}\textbf{PRINT}\\ \textbf{BLANK}\\ \textbf{SUPPRESS}\end{array}\right\}$	Turns on the automatic incremented counter.
MODE NCKPAR	Displays all characters as if they had parity on (default).
MODE NCOLUMN	Turns off the column display (default).
MODE NCOUNT	Suspends counter incrementing (default).
MODE NUMBER	Displays line numbers in front of the displayed line.
MODE NNUMBER	Turns off the line number display (default).
MODE NOSEMI	Turns off the line terminator function of semicolons, allowing semicolons to be used as a regular print character in INPUT mode.

MODE PRALL	Displays lowercase characters if the device has that capability.
MODE PRUPPER	Displays all characters as uppercase. Precedes lowercase characters with a ^L and precedes uppercase characters with a ^U if the device is uppercase only.
MODE PROMPT	Displays prompt characters for INPUT and EDIT modes.
MODE NPROMPT	Stops displaying of INPUT and EDIT prompt characters (default).
MODE SEMI	Uses semicolons as line terminators (default).
MODIFY/*string-2*/*string-1*/[G][*n*]	Superimposes *string-1* onto *string-2* for *n* lines. If G is omitted, only the first occurrence of *string-1* on each line is modified; otherwise, all occurrences of *string-1* are modified.
MOVE *buffer-1* $\left\{\begin{array}{l} buffer\text{-}2 \\ /string/ \end{array}\right\}$	Moves *string* or contents of *buffer-2* into *buffer-1*.
NEXT [*n*]	Moves the pointer *n* lines forward or backward (default *n* = 1).
NFIND *string*	Moves the pointer down to the first line not beginning with *string*.
NFIND(*n*) *string*	Moves the pointer down to the first line in which *string* does not start in column *n*.
NLOCATE *string*	Finds the first line that does not contain *string* anywhere in the line.
OOPS	Undoes the last line changed and returns it to its status before the modification.
OVERLAY *string*	Superimposes *string* on the current line. Use tabs to start in the middle of the line. Use ! to delete existing characters. (A blank in the string leaves the old character in place.)
PAUSE	Returns temporarily to PRIMOS for the use of PRIMOS-level commands. START returns to the previous ED position.
POINT *line-number*	Relocates the pointer to *line-number*.
PPRINT [*first*] [*last*]	The PPRINT command displays a range of lines relative to the current position without changing the current position: *first* — Number of lines away from current position to start displaying *last* — Number of lines away from current position to stop displaying If only one *positive* number is specified, it is interpreted as the ending-line position (*last*) and the default starting line is the current line.

	If only one *negative* number is specified, it is interpreted as the beginning-line position (*first*) and the default ending line is the current line. If no numbers are given, the default is PP –5 5, which displays from five lines above to five lines below the current position.
PRINT [*n*]	Displays the current line or *n* lines beginning with the current line. Moves the pointer to the last line displayed.
PSYMBOL	Displays a list of current symbol characters and their function.
PTABSET *tab-1...tab-8*	Provides for a setup of tabs on devices that have physical tab stops.
PUNCH { **ASR**[*n*] / **PTP** }	Punches *n* lines on high-speed or low-speed paper-tape punch.
QUIT	Returns control to PRIMOS without filing text. If file has been modified, ED warns user and asks: `OK TO QUIT?`
QF	The QUIT FINAL command lets the user QUIT out of a modified file without having ED ask if it may throw away the work file.
RETYPE *string*	The current line is replaced by *string*.
SAVE [*filename*]	Saves file without leaving ED. If user does not specify *filename*, ED saves into the file being edited and displays its name.
SYMBOL *name character*	Changes a symbol *name* to *character*. Current default values are

Name	*Default Characters*
KILL	?
ERASE	"
WILD	!
BLANK	#
TAB	\
ESCAPE	^
SEMICO	;
CPROMPT	$
DPROMPT	&

TABSET *tab-1...tab-8*	Sets as many as eight logical tab stops to be invoked by the tab symbol (\).
TOP	Moves the pointer to a null line before the first line of text.
UNLOAD *filename* [*n*]	Copies *n* lines into *filename*.

UNLOAD *filename* TO *string*	Unloads lines from current file into *filename* until *string* is found.
VERIFY	Displays each line after completion of certain commands (default).
WHERE	Displays the current line number.
XEQ [*buffer*]	Executes the contents of *buffer*. If no buffer name is given, the last command line is reexecuted.
*[*n*]	Causes the preceding command to be repeated *n* times as in `F /;D;*10` which deletes the next ten lines that begin with / . If *n* is omitted, the command repeats until the bottom of the file is reached.

ED Defaults

INPUT (TTY)

LINESZ 144

MODE NCKPAR

MODE NCOLUMN

MODE NCOUNT

MODE NNUMBER

MODE NPROMPT

MODE PRALL

VERIFY

ED Symbols

TABLE E-1
ED Symbols

Character	*Name*	*Interpretation*
#	BLANKS	Match *n* spaces (with FIND, NFIND)
@	COUNTER	MODE COUNT's counter symbol
$	CPROMPT	MODE PROMPT's edit prompt
&	DPROMPT	MODE PROMPT's input prompt
"	ERASE	Character erase
^	ESCAPE	Logical escape (see Notes below)
?	KILL	Line kill
;	SEMICO	End of line or command
\	TAB	Logical tab
!	WILD	Match any character (with FIND, NFIND)

Notes

Users may change any of these special characters from within the ED editor, with the SYMBOL command.

Nonprinting characters may be entered into text with the ED editor by using the logical escape character (^ is the default escape character) and the octal value. The nonprinting character is interpreted by output devices according to their hardware. For example, typing ^207 (where ^ = escape) enters *one* character into the text.

F

Directory Passwords

Directory passwords provide an alternative to Access Control Lists (ACLs) as a protection system for controlling the use of files and directories. This appendix describes how to use the password system. In particular, it describes how to

- Assign passwords to directories (PASSWD)
- Use passwords to gain access to directories
- Set specific access rights on file system objects (PROTECT)
- Convert a password directory to an ACL directory (SET_ACCESS)
- Convert an ACL directory to a password directory (REVERT_PASSWORD)
- Create a password subdirectory under an ACL directory (CREATE)

Note

Because ACLs are more flexible than passwords and offer greater protection, most systems use ACL protection. The information in this appendix is provided primarily for users of systems where passwords have been in use, and where the change to ACLs has not happened or is not yet complete.

Assigning Directory Passwords

You can secure your directories against unauthorized users by assigning passwords with the PASSWD command. The two levels of passwords are owner and non-owner. If you give the owner password in an ATTACH command, you have owner status; if you give the non-owner password in an ATTACH command, you have non-owner status. If you fail to give a required password correctly in an ATTACH command, PRIMOS returns the message `Bad password` and does not execute the ATTACH command. File system objects within a password-protected directory can be given different access rights for owners and non-owners with the PROTECT command. (See Setting Access Rights on File System Objects, below.)

The PASSWD command either replaces any existing password(s) on the current directory with one or two new passwords or assigns passwords to this directory if there are none. The format is

PASSWD [*owner-password* [*non-owner-password*]]

The *owner-password* is specified first; the *non-owner-password* follows.

For example, assume the current directory has no password:

```
OK, PASSWD US THEM
OK, PASSWD NIX TRIX
OK,
```

The first command line sets the owner password US and the non-owner password THEM on the current directory. The second command line changes the owner password from US to NIX and the non-owner password from THEM to TRIX.

Passwords may contain almost any characters, but they may not begin with a digit (0-9).

If an *owner-password* is specified and a *non-owner-password* is not specified, the default for the non-owner password is null; then, any password (except the owner password) or none allows access to this directory as a non-owner.

If the PASSWD command is given alone, without either an *owner-password* or a *non-owner-password*, then the *owner-password* is set to blanks and the *non-owner-password* is set to null. Specifying no password gives access to the directory as an owner; specifying any password gives access to the directory as a non-owner. The PASSWD command given alone is useful when you wish to remove passwords from a directory.

You must have owner status on a directory in order to use the PASSWD command.

Using Passwords to Gain Access to Directories

If a directory has a password, the password must be given whenever the directory name is given. The password is entered after the name of the directory, separated by one blank space. For example, assume ROCKET is the owner password for the directory JONES. The command

```
OK, ATTACH JONES ROCKET
```

connects you to JONES with owner status.

If a password is used in a pathname, apostrophes enclose the entire pathname. For example,

```
OK, ATTACH 'MAPLE HIDDEN>BRANCH4'
```

Setting Access Rights on File System Objects

Specific access rights to the objects (files, segment directories, and subdirectories) within a password-protected directory can be established by using the PROTECT command. This command sets the protection codes for users with owner and non-owner status in the directory. The format is

PROTECT ***pathname*** [*owner-rights*] [*non-owner-rights*] [**–REPORT**]

Argument/Option	*Meaning*
pathname	The name of the object to be protected
owner-rights	A code specifying owner's access rights to the object
non-owner-rights	A code specifying the non-owner's access rights to the object
–REPORT	An option specifying that the results of executing the command be reported to the user

The values and meanings of the protection codes are

Code	*Rights*
NIL	No access of any kind allowed
R	Read only
W	Write only
D	Delete only
RW	Read and Write
RD	Read and Delete
WD	Write and Delete
RWD	Read, Write, and Delete (All access)

To use the PROTECT command on objects in a directory, you must have owner status on the directory. For example,

```
OK, PROTECT MYDIR>LETTER RWD R
```

In this example, protection codes are set on the file LETTER in the directory MYDIR so that All access rights are given to the owner and only Read access is given to the non-owner.

Notes

The default access codes associated with any newly created object are RWD NIL. The owner is given All rights and the non-owner is given None. Default values for the PROTECT command, however, are NIL NIL. Thus, the command PROTECT MYFILE denies All rights to owner and non-owner alike. The owner can always recover from this situation because he can change the protection codes again and grant himself All access rights.

Although the PROTECT command may be used to modify the protection codes of objects in ACL directories, the ACL mechanism takes precedence and the codes are ignored when the object is accessed. If the ACL directory is converted to a password directory, the protection codes establish their designated owner and non-owner access rights. To use the PROTECT command for objects in an ACL directory, you must have Protect (P) access to the directory.

Converting a Password Directory to an ACL Directory

Conversion from a password directory to an ACL directory is done automatically whenever the SET_ACCESS command (explained in Chapter 5) is given on a password directory whose parent is an ACL directory. The command does not convert a password directory whose parent is a password directory.

To convert a directory, you must either have Protect rights to the parent directory or there must be no owner password in the directory being converted.

Note

To convert a directory with passwords when you do not have Protect access to the parent, give this sequence of commands:

> **ATTACH** ***directory-pathname password***
> **PASSWD**
> **SET_ACCESS** ***directory-pathname acl***

If you do not have Protect access to the parent directory, you must supply an ACL with the SET_ACCESS command, giving yourself access rights, or you lose access to the directory after it is converted. PRIMOS warns you if this situation is about to occur, and allows you to abort the SET_ACCESS command.

Examples

In example one, you wish to convert the password directory HAND (with the pathname GLOVE>HAND) to an ACL directory with default ACL protection. GLOVE is an ACL directory. If you are attached to GLOVE, give the command

```
OK, SET_ACCESS HAND
```

If you are attached elsewhere, give the command

```
OK, SET_ACCESS GLOVE>HAND
```

PRIMOS responds with the `OK,` prompt, and HAND is now protected by the same ACL that protects GLOVE.

In example two, you wish to convert the password directory FOOT (with the pathname SHOE>SOCK>FOOT) to an ACL directory with the same protection existing on SHOE. SHOE is an ACL directory, but SOCK is a password directory. The command

```
OK, SET_ACCESS SHOE>SOCK>FOOT
```

fails because the parent directory of FOOT is also a password directory. To convert FOOT, you must first convert SOCK and then convert FOOT.

In example three, to convert an entire subtree, you may use the wildcard and treewalking capabilities of PRIMOS, explained in Chapter 6:

```
OK, SET_ACCESS  *>@>@  -WALK_FROM 1 -DIR
```

Converting an ACL Directory to a Password Directory

To convert an ACL directory to a password directory, use the command

```
OK, REVERT_PASSWORD
```

The command takes no arguments and converts the current directory only. You must therefore be attached to the directory you wish to convert. The following constraints also apply:

- You must have Protect (P) access to the directory before you convert it.
- When you convert a directory, its original password(s) and protection codes are reactivated.
- You may not have ACL-protected subdirectories under a password directory. Therefore, if you give the REVERT_PASSWORD command for a directory that contains ACL-protected subdirectories or any access categories, the command fails.

Note

If you convert an ACL directory to a password directory or vice versa and wish to use the LD or LIST_ACCESS command to check the conversion, you must reattach to the directory for the changes to take effect. Access rights are calculated when you first attach to the directory, and the access is not recalculated to reflect the change until you attach somewhere else. If you convert your origin directory and subsequently use the ORIGIN command, the directory is not converted. For the ORIGIN command to reflect the conversion, you must log out and then log in again.

Example

Consider the subtree SHIRT>SLEEVE>CUFF. You wish to convert ACL directory SLEEVE to a password directory. CUFF is also an ACL directory. If you attach to SLEEVE and give the REVERT_PASSWORD command, the command fails. You get the error message at the end of the following sequence:

```
OK, ATTACH SHIRT>SLEEVE
OK, REVERT_PASSWORD
Directory still contains ACL subdirectories.  <Current directory>
     (revert_password)
ER!
```

To convert SLEEVE to a password directory, you must first convert CUFF to a password directory. Then you can convert SLEEVE (because SLEEVE no longer contains any ACL subdirectories). The following sequence illustrates the conversion:

```
OK, ATTACH SHIRT>SLEEVE>CUFF
OK, REVERT_PASSWORD
OK, ATTACH SHIRT>SLEEVE
OK, REVERT_PASSWORD
OK,
```

Creating a Password Subdirectory Under an ACL Directory

The CREATE command allows the creation of a password subdirectory under an ACL directory. You must have Add (A) access to the parent directory to create a new subdirectory. The format is

CREATE ***pathname*** **[–PASSWORD]**

Specifying –PASSWORD creates a password subdirectory. If the option is omitted, the CREATE command creates a subdirectory of the same type as its parent directory.

You cannot create an ACL directory under a password directory.

G System Information

PRIMOS supports many commands that provide useful information about the availability and current usage of system resources. The following table summarizes the information available and the commands to obtain it. All of these commands are fully explained in the *PRIMOS Commands Reference Guide.*

Further commands, called System Information and Metering (SIM) commands, may be available if your System Administrator has established user access to them. These commands are fully documented in the *DSM User's Guide.*

Note

The STATUS ALL command gives information provided by all of the STATUS commands listed below (except STATUS PROJECTS). The STATUS USERS command gives a wide variety of information about user IDs, user numbers, and line and device assignments on your system.

Information on	*PRIMOS Commands*	*Use*
ACLs	LIST_ACCESS [*pathname*]	Lists access rights to a file system object
	LIST_GROUP	Lists your user groups
Batch	BATCH –STATUS	Shows number of waiting, deferred, or held jobs
	BATCH –DISPLAY	Lists user IDs, job IDs, and queue numbers
	JOB –STATUS	Lists Batch jobs for your user ID
	JOB –DISPLAY	Displays detailed information on your Batch jobs
	BATGEN –STATUS	Lists available batch queues

Information on	*PRIMOS Commands*	*Use*
	BATGEN –DISPLAY	Displays detailed information on the batch queues
Command Environment	LIST_LIMITS	Shows your command environment limits
	LIST_EPF	Lists EPFs mapped to your address space
	LIST_SEGMENT	Lists segments you are using and shows your access
Disks	STATUS DISKS	Shows disks visible on your system
	AVAIL [*diskname*]	Shows space available on a logical disk
File System	LIST_QUOTA [*pathname*]	Shows directory quota and number of records used
	STATUS UNITS	Shows file-units you have open
	LD –SIZE SIZE *pathname*	Show sizes of files
File Transfer	FTR –STATUS	Lists your transfers
	FTR –DISPLAY	Displays detailed information on each transfer
Global Variables	LIST_VAR	Lists global variables in active global variable file
Messages	MESSAGE –STATUS	Shows receive states
Network	STATUS NETWORK	Tells if network is available and what systems are configured
	LIST_REMOTE_ID	Lists the remote IDs you have added with ADD_REMOTE_ID
Printers	PROP –STATUS	Lists environment names for available printers
	PROP [*environment-name*] –DISPLAY	Displays detailed information on a specific printer
Projects	STATUS PROJECTS	Lists users and their project names
Spool	SPOOL –LIST	Shows contents of spool queue

Information on	*PRIMOS Commands*	*Use*
Terminal Settings	TERM –DISPLAY	Shows current terminal settings
Time and Date	DATE	Shows current date and time
	TIME	Shows your usage of system resources
Tape Drives	STATUS DEVICE	Lists logical and physical device numbers of tape units assigned to users
Users	STATUS USERS	Shows IDs of all users on the system along with user numbers and assigned lines and devices
	STATUS ME	Shows the same information as STATUS USERS, but for your user ID only

Index

Index

Symbols

" character, 1-8
$$ JOB, 16-12
? character, 1-8
~, syntax suppression character, 6-13

Numbers

32I mode, 10-5
32IX mode, 10-6
64V mode, 10-5

A

A access, 5-26
ABBREV command, 8-4
  ADD option, 8-6
  ADD_ARGUMENT option, 8-6
  ADD_COMMAND option, 8-5
  CREATE option, 8-4
  DELETE option, 8-8
  format, 8-4
  LIST option, 8-9
  OFF option, 8-8
  ON option, 8-8
Abbreviation files
  activating, 8-8
  creating, 8-4
Abbreviations
  defining, 8-5
  deleting, 8-8
  global variables in, 8-12
  listing, 8-9
  multiple commands, 8-6
  using, 8-9
  variables in, 8-7
ABORT, option of JOB command, 16-12
Absolute pathnames, 2-6
ACCEPT, option of MESSAGE command, 8-17
Access categories, 2-2, 5-2, 5-9
  creating, 5-11
  deleting, 5-14
  listing contents, 5-10
  modifying, 5-13
  protecting objects with, 5-10
  replacing the contents of, 5-12
  symbol, 2-2
Access control list
  see: ACLS
Access rights, 5-1
  Add (A), 5-26
  ALL, 5-27
  Delete (D), 5-24
  Execute (X), 5-27
  in password-protected directories, F-2
  List (L), 5-26
  list of, 5-2
  listing, 5-3
  matching, 5-20
  NONE, 5-27
  overlapping, 5-5
  Owner (O), 5-23
  Protect (P), 5-23
  Read (R), 5-26
  specifying, 5-4
  types of, 5-22
  Use (U), 5-26
  with FTS, 19-13
  Write (W), 5-27
ACCOUNTING, option of JOB command, 16-7
ACLs, 5-1
  access categories, 5-2
  converting to password protection, F-5
  creating, 5-6
  default protection, 5-2, 5-14
  default protection with access categories, 5-17
  default protection with specific ACLs, 5-15
  DELETE command, 5-14
  EDIT_ACCESS command, 5-8, 5-13
  group names, 5-4
  LIST_ACCESS command, 5-3, 5-9, 5-15
  listing access rights, 5-3
  matching access rights, 5-20
  modifying, 5-8
  on remote systems, 19-4, 19-6, 19-9
  overlapping access rights, 5-5
  priority, 5-28
  $REST, 5-4
  SET_ACCESS command, 5-6, 5-11 to 5-12, 5-18, 5-20
  setting, 5-2
  specific, 5-2, 5-5
  specifying rights, 5-4
  types of users, 5-4
  who can set, 5-20
  see also: Access rights
Active window, 16-18
Add access, 5-26
ADD, option of ABBREV command, 8-6
ADD_ARGUMENT, option of ABBREV command, 8-6
ADD_COMMAND, option of ABBREV command, 8-5
ADD_REMOTE_ID command, 19-5
  in LOGIN.CPL, 19-6
  PROMPT option, 19-5 to 19-6
Addressing modes, 9-3, 10-5
ALIAS, option of UNASSIGN command, 18-4

ALL access, 5-27
ALL, option of CLOSE command, 3-16, 14-3
APPEND, ED command, 4-16
&ARGS, CPL directive, 15-5
ARGS, option of JOB command, 16-8
ASCII files, 17-1
ASSIGN command
  ALIAS argument, 18-2
  error messages, 18-3
  messages, 18-2
  MT argument, 18-2
  MTX argument, 18-2
  WAIT option, 18-2
ATTACH command, 3-3, F-1
Attaching to directories, 3-3
ATTRIBUTE, option of SPOOL command, 4-27
Auto Speed Detect, 1-5

B

Backing up files, 18-5
Basenames, 2-3
BATCH command
  DISPLAY option, 16-16
  STATUS option, 16-16
Batch jobs
  $$ JOB command line, 16-12
  aborting, 16-12
  active, 16-10
  canceling, 16-12
  creating, 16-6
  job IDs, 16-10
  jobnames, 16-10
  modifying and canceling, 16-10
  modifying options, 16-10
  monitoring, 16-13
  submitting, 16-7
Batch processing, 1-2
Batch queues
  active window, 16-18
  cptime, 16-18
  delta rlevel, 16-19
  etime, 16-18
  funit, 16-19
  monitoring, 16-17
  names, 16-18
  priority, 16-19
  timeslice, 16-19
Batch system
  Batch Administrator, 16-6
  Batch monitor, 16-6
  Batch queues, 16-6
  JOB command, 16-7
  job files, 16-5
  job ID, 16-7
  monitoring, 16-16
  queues, monitoring, 16-17
BATGEN command
  DISPLAY option, 16-17
  STATUS option, 16-17
BINARY, compiler option, 10-4
Binary files
  see: Object files
BIND command, 9-3, 11-1, 11-3
  errors, 11-4
  examples, 11-6
  FILE subcommand, 11-5
  HELP subcommand, 11-5
  interactive use, 11-4
  invoking, 11-3
  libraries, 11-6
  LIBRARY subcommand, 11-4
  LOAD subcommand, 11-4
  MAP subcommand, 11-5
  normal linking with, 11-5
  options, 11-4
  order of linking, 11-6
  QUIT subcommand, 11-5
  subcommands, 11-4
BOTTOM, ED command, 4-12
BOTTOM_UP, treewalking option, 6-10
BREAK, option of TERM command, 1-15
BREAKPOINT, DBG subcommand, 13-8
BRIEF, option of CMPF command, 17-7
BRIEF, option of SORT command, 17-2

C

C, subcommand of NETLINK, 19-10
CANCEL, option of FTR command, 19-21
CANCEL, option of JOB command, 16-12
CANCEL, option of SPOOL command, 4-26
CHANGE, ED command, 4-16
CHANGE, option of JOB command, 16-10
CHANGE_PASSWORD command, 1-14
Character data, 4-1
CLEAR, DBG subcommand, 13-9
CLOSE command, 3-16, 14-2
  ALL option, 14-3
CMPF command, 17-6
  options, list of, 17-7
CNAME command, 3-7
COMI files
  see: Command input files
COMINPUT command, 14-5
  CONTINUE option, 14-7
  options, list of, 14-6
COMINPUT files
  see: Command input files
Command environment, 12-5
  breadth, 12-7
  checking limits, 12-7
  depth, 12-6
  limits, 12-6
Command files, 14-1
Command input files, 1-2, 4-1, 14-4
  chaining, 14-7
  closing, 14-9
  comments, 14-5
  ending, 14-5
  errors, 14-7
  example, 14-6
  file-units in, 14-8
  naming, 14-5
  phantom execution, 16-2
Command level, 1-7, 8-2, 12-6
Command Line Editor
  see: ECL
Command line features, 6-1
  iteration, 6-2
  multiple commands, 6-1
  name generation, 6-10
  treewalking, 6-8
  wildcards, 6-2
Command lines
  arguments, 1-7
  arguments in CPL programs, 15-5 to 15-6
  format, 1-7
  modifying and editing, 1-8
  options, 1-7
Command output files
  see: COMO files
Command Procedure Language
  see: CPL
Command processor, 1-7
Commands, interrupting, 3-15
COMO files, 3-14, 14-1
  closing, 14-2
  creating, 14-2
  dating, 14-4
  naming, 14-2
  opening from phantoms, 16-2
COMOUTPUT command, 3-14, 14-2
  END option, 3-14, 14-2
  options, list of, 14-3
COMOUTPUT, option of JOB command, 16-8

Comparing files, 17-6
Compatibility, 1-3
Compiler name suffixes, list of, 10-3
Compilers, 9-2
  addressing modes, list of, 10-6
  BINARY option, 10-4
  cross-reference listings, 10-7
  DEBUG option, 13-4
  defaults, 10-4
  invoking, 10-2
  list of, 10-2
  listing files, 10-6
  LISTING option, 10-7
  messages, 10-7
  XREF option, 10-7
Compiling programs, 9-2, 10-1
CONCAT command, 17-10
Condition mechanism, 20-1
Conditions, list of, 20-1
CONNECT, NTS command, 1-13
CONTINUE, DBG subcommand, 13-8
CONTINUE, option of COMINPUT command, 14-6 to 14-7
CONTINUE, option of COMOUTPUT command, 14-3
Control key characters, 1-9
COPIES, option of SPOOL command, 4-26
COPY command, 3-8, 19-4
  illustrations, 3-9
Copying remote files, 19-4
CPL, 1-2
  &ARGS directive, 15-5
  arithmetic operators, 15-9
  assignment operator, 15-7
  command line arguments, 15-5 to 15-6
  &DATA directive, 15-13, 15-16
  debugging, 15-16
  decision making, 15-8
  directives, 15-1
  directives, summary of, 15-17
  &DO directive, 15-11
  &ELSE directive, 15-10
  &END directive, 15-11, 15-14
  ending programs, 15-15
  errors, 15-16
  EXISTS function, 15-12
  function calls, 15-12
  global variables, 15-8
  &IF directive, 15-8 to 15-10
  interpreter, 15-1
  invoking programs, 15-2
  logical operators, 15-9
  naming files, 15-2
  NULL function, 15-12
  null strings, 15-7
  overview, 15-1
  phantom execution of programs, 16-2
  PRIMOS commands in programs, 15-3
  programming, 15-2
  relational operators, 15-9
  &RETURN directive, 15-15
  &SET_VAR directive, 15-7
  &SEVERITY directive, 15-16
  subsystems with, 15-13
  terminal input, 15-14
  &TTY directive, 15-14
  variables, 15-4
  variables, defining and referencing, 15-6
CPL command, 15-2
CPL, option of JOB command, 16-8
Cptime, 16-18
CPTIME, option of JOB command, 16-8
CREATE command, 3-5
CREATE, option of ABBREV command, 8-4
Creating new directories, 3-5
Cross-reference listings, 10-7
Ctrl-P, 1-9
Ctrl-Q, 1-9
Ctrl-S, 1-9
Current directory, 2-9
Current disk, 2-9
Customizing your environment, 8-1

D

D access, 5-24
Data, character, 4-1
&DATA, CPL directive, 15-13, 15-16
DATE command, 14-4
DBG, 9-3, 13-1
  BREAKPOINT subcommand, 13-8
  Breakpointing, 13-8
  CLEAR subcommand, 13-9
  clearing breakpoints, 13-9
  code requirements, 13-1
  compiling and linking programs, 13-4
  CONTINUE subcommand, 13-8
  data manipulation features, 13-3
  evaluate subcommand (:), 13-10
  examining and modifying data, 13-10
  examining source code, 13-5
  invoking, 13-5
  LET subcommand, 13-10
  LIST subcommand, 13-9
  listing breakpoints, 13-9
  miscellaneous features, 13-4
  program control features, 13-2
  QUIT subcommand, 13-5
  RESTART subcommand, 13-8
  sample session, 13-11
  SOURCE subcommand, 13-5
  SOURCE subcommand arguments, 13-5
  SOURCE subcommand line numbers, 13-7
  starting program execution, 13-8
  STEP subcommand, 13-9
  stepping, 13-9
  stopping execution, 13-8
  terminating a session, 13-5
  tracing features, 13-3
  WATCH subcommand, 13-10
DC, LOAD subcommand, 11-9
DEBUG, compiler option, 13-4
Debugging CPL programs, 15-16
Debugging programs, 9-3, 13-1
  see also: DBG
Default protection, 5-2, 5-14
  listing, 5-15
  returning to, 5-18
  with access categories, 5-17
  with specific ACLs, 5-15
DEFER, option of FTR command, 19-16
DEFER, option of JOB command, 16-8
DEFER, option of MESSAGE command, 8-17
DEFER, option of SPOOL command, 4-27
DEFINE_GVAR command, 8-11
Delete access, 5-24
DELETE command, 3-11, 5-14
DELETE, ED command, 4-17
DELETE, option of ABBREV command, 8-8
Delete protection
  determining, 3-13
  setting, 3-12
DELETE_VAR command, 8-11
Delta rlevel, 16-19
DEVICE LP, option of FTR command, 19-16
Dialup terminals, 1-5
Directories, 2-2
  attaching to, 2-9
  creating, 3-5
  current, 2-9
  MFDs, 2-6
  origin, 2-9
  parent, 2-6
  password-protected, 2-7
  passwords, F-1
  subdirectories, 2-6
  symbol, 2-2
  top-level, 2-6

Directory contents, examining, 3-1
Directory passwords
  assigning, F-1
  in pathnames, F-2
  non-owner, F-1
  owner, F-1
Disk numbers, 2-8
Disk quotas, 8-18
Disk volumes, 2-6
Disks, 2-6
  adding, 19-2
  availability of, 19-3
  current, 2-9
  numbers, 2-8
DISPLAY, option of BATCH command, 16-16
DISPLAY, option of BATGEN command, 16-17
DISPLAY, option of FTR command, 19-18
DISPLAY, option of JOB command, 16-14
DISPLAY, option of TERM command, 1-16
Display, starting and stopping, 1-9
&DO, CPL directive, 15-11
Document processing, 4-4
DSTN_NTFY, option of FTR command, 19-20
DSTN_SITE, option of FTR command, 19-15
DSTN_USER, option of FTR command, 19-16, 19-20
DUNLOAD, ED command, 4-21
Dynamic segments, 12-7

E

ECL, 1-8 to 1-9
  aborting commands, 7-7
  changing case and character position, 7-9
  command conventions, 7-2
  command history, 7-6, 7-11
  command macros, 7-20
  command summary, 7-24
  defining, copying, and deleting, 7-10
  deleting characters and lines, 7-5
  expanding abbreviations, 7-20
  expanding pathnames, 7-16
  explaining keys, 7-20
  extended commands, 7-24
  hidden commands, 7-14
  moving and deleting words, 7-8
  moving the cursor, 7-3
  network, using across, 7-24
  numeric arguments, 7-20
  password command, 7-23
  prompts, 7-23
  redisplaying previous command line, 7-5
  referencing other directories, 7-18
  refreshing current command line, 7-5
  repeating commands, 7-8
  restoring copied and deleted text, 7-11
  searching, 7-6
  starting and stopping, 7-2
ECL command, 7-2
ECL options
  ENTRY, 7-9
  ERROR_BRIEF, 7-23
  list, 7-28
  OFF, 7-2
  ON, 7-2
  READY_BRIEF, 7-23
  STACK, 7-15
  STICK, 7-14
  WARNING_BRIEF, 7-23
  WILD_TAIL, 7-17
ED, 4-2, 10-1
  command arguments, 4-9
  command format, 4-9
  copying text to and from disk, 4-20
  current line, 4-8
  EDIT mode, 4-6
  empty lines, 4-6
  entering text, 4-7
  error messages, 4-10
  escape character, 4-6
  INPUT mode, 4-6
  invoking, 4-5
  line numbers, 4-12
  modes, 4-5
  modifying text, 4-16
  moving pointer, 4-11
  multiple commands, 4-10
  null lines, 4-8
  quitting and saving, 4-22
  sample file, 4-7
  searching, 4-13
  special characters, 4-6
  switching modes, 4-6
  table of basic commands, 4-11
ED command, 4-5
ED commands
  APPEND, 4-16
  arguments to, 4-9
  BOTTOM, 4-12
  CHANGE, 4-16
  DELETE, 4-17
  DUNLOAD, 4-21
  FILE, 4-23
  FIND, 4-14
  format of, 4-9
  IB, 4-18
  INSERT, 4-18
  LOAD, 4-22
  LOCATE, 4-13
  multiple, on line, 4-10
  NEXT, 4-12
  NFIND, 4-14
  OOPS, 4-19
  POINT, 4-13
  PRINT, 4-10
  QUIT, 4-22
  RETYPE, 4-18
  SAVE, 4-23
  table of, 4-11
  TOP, 4-11
  UNLOAD, 4-20
  WHERE, 4-12
EDIT_ACCESS command, 5-8, 5-13
  NO_QUERY option, 5-8
EDIT_CMD_LINE
  see: ECL
Editing process, 4-2
EDITOR
  see: ED
&ELSE, CPL directive, 15-10
EMACS, 4-2, 10-1
  language modes, 10-1
  with NETLINK, 19-12
&END, CPL directive, 15-11, 15-14
END, option of COMINPUT command, 14-6
END, option of COMOUTPUT command, 3-14, 14-2 to 14-3
ENTRY, option of ECL command, 7-9
Entryname
  see: Objectnames
EPFs, 9-3, 11-1, 12-6
  active, 12-6
  advantages of, 11-2
  inactive, 12-6
  restarting, 12-4
  suspended, 12-6
ER! prompt, 1-6, 8-2
Erase character, 1-8
ERASE, option of TERM command, 1-15
Error messages
  ASSIGN command, 18-3
  compiler, 10-7
  list of, D-1
  runtime, 12-5
Error prompts, 8-3
ERROR_BRIEF, option of ECL

command, 7-23
Errors
ATTACH, 3-4
in command input files, 14-7
in CPL programs, 15-16
remote file access, 19-6
Etime, 16-18
ETIME, option of JOB command, 16-8
Evaluate (:), DBG subcommand, 13-10
Executable files
see: Runfiles
Executable Program Formats
see: EPFs
Execute access, 5-27
EXISTS function, 15-12

F

FILE, BIND subcommand, 11-5
FILE, ED command, 4-23
File system, 2-1
commands, 3-1
objects, 2-1
password protection, F-1
quotas, 8-18
symbols for objects, 2-2
tree structure, 2-2, 2-5
File system objects, 2-1
naming, 2-3
storage of, 2-3
symbols for, 2-2
see also: Files
File Transfer Service
see: FTS
File tree, 2-5
File tree diagram, 2-5
Filename, COMO files, 14-2
Filenames, 2-1
basenames, 2-3
characters in, 2-3
command input files, 14-5
CPL, 15-2
object, 10-4
runfiles, 12-3
source files, 10-2
suffix conventions, 2-4
suffixes, 2-3
Files, 2-1
ASCII, 17-1
backing up, 18-5
command, 14-1
command input, 4-1, 14-4
COMO, 3-14, 14-1
comparing, 17-6
concatenating, 17-10
copying, 3-8
deleting, 3-11
job, 16-5
listing, 3-6
merging, 17-8
object, 9-2
printing, 4-24
program listing, 10-6
protecting, 5-1 to 5-2
protecting from deletion, 3-12
remote access, 19-2
remote, ACLs, 19-4, 19-6, 19-9
remote, copying, 19-4
renaming, 3-7
restoring from tape, 18-6
runfiles, 9-2, 11-1
sorting, 17-1
source, 10-1
symbol, 2-2
text, 4-1
transferring with FTS, 19-13
transparent access, 1-3
File-units, 14-7, 16-2
FILMEM command, 11-9
FIND, ED command, 4-14
FORCE, option of MRGF command, 17-8
Forced user validation, 19-5
FROM, option of SPOOL command, 4-28
FTR command, 19-14
CANCEL option, 19-22
DEFER option, 19-16
DEVICE LP option, 19-16
DISPLAY option, 19-18
DSTN_NTFY option, 19-20
DSTN_SITE option, 19-15
DSTN_USER option, 19-16, 19-20
HELP option, 19-21
LOG option, 19-19
PRIORITY option, 19-16
RELEASE option, 19-22
SOURCE_USER option, 19-20
SRC_NTFY option, 19-20
SRC_SITE option, 19-15
STATUS option, 19-17
FTS, 19-13
access rights, 19-13
canceling requests, 19-21
checking status of requests, 19-17
deferring transfer, 19-16
FTR command, 19-16
help facility, 19-21
logging request events, 19-19
printing a file, 19-16
receiving files, 19-15
request names, 19-14
request numbers, 19-14
request on hold, 19-22
requesting notification, 19-20
sending files, 19-14
sites, 19-14
temporary destination files, 19-14
when to use, 19-22
Full pathnames, 2-8
Function calls, 15-12
FUNIT, option of JOB command, 16-8

G

Global variable files, 8-10
Global variables, 8-10
commands, 8-10
in abbreviations, 8-12
in Batch and phantom environments, 8-13
in CPL programs, 15-8
in interactive commands, 8-11
in phantoms, 16-3
in programs, 8-13
GO, subcommand of MRGF, 17-8
Group names in ACLs, 5-4

H

HELP, BIND subcommand, 11-5
HELP command, 1-16
HELP, option of FTR command, 19-21
HELP, subcommand of NETLINK, 19-12
HOME, option of JOB command, 16-9

I

I mode, 10-5
IAP
see: Initial Attach Point
IB, ED command, 4-18
IDs, project, 1-12
IDs, user, 1-10
&IF, CPL directive, 15-8 to 15-10
Information, online, 1-16
Information, requesting with HELP, 1-16
Initial Attach Point (IAP), 2-9
INITIALIZE, LOAD subcommand, 11-9
INITIALIZE_COMMAND_ENVIRONMENT command, 12-9 to 12-10
INSERT, ED command, 4-18
Interactive dialog, 1-2

Interrupting commands, 3-15
Invoking programs, 12-2
Iteration, 6-2

J

Job command, 16-7
JOB command
  ABORT option, 16-12
  CANCEL option, 16-12
  CHANGE option, 16-10
  DISPLAY option, 16-14
  manage-options, 16-10
  monitor-options, 16-14
  NOTIFY option, 16-14
  options, 16-7
  PRIORITY option, 16-11
  QUEUE option, 16-11
  RESTART option, 16-11
  STATUS option, 16-14
  submit-options, 16-7, 16-11
Joining files, 17-10

K

Kill character, 1-8
KILL, option of TERM command, 1-15

L

L access, 5-26
LAN Terminal Server, 1-5
LAN300, 1-3, 1-5, 19-1
LANs, 1-3
  diagram, 1-4
  LAN Terminal Server, 1-5
  LAN300, 1-3, 1-5, 19-1
  NTS, 1-5
  RINGNET, 19-1
Leading command, 3-1
  arguments and options, 3-2
  PROTECT option, 3-13
  using wildcards with, 6-6
LET, DBG subcommand, 13-10
Libraries, linking, 11-3
LIBRARY, BIND subcommand, 11-4
LIBRARY, LOAD subcommand, 11-9
Library names, list of, 11-3
LIKE, option of SET_ACCESS command, 5-20
Linking process, 11-2
  external references, 11-2
  libraries, 11-2
  user-supplied subroutines, 11-2
Linking programs, 9-2, 11-1
Linking utilities, 9-3
  addressing modes, 11-2
List access, 5-26
LIST, DBG subcommand, 13-9
LIST, option of ABBREV command, 8-9
LIST, option of SPOOL command, 4-25
LIST_ACCESS command, 5-3, 5-9, 5-15, 5-28
LIST_EPF command, 12-9
LISTING, compiler option, 10-7
LIST_LIMITS command, 12-7, 12-9
LIST_MINI_COMMANDS command, 12-9, 12-12
LIST_PRIORITY_ACCESS command, 5-28
LIST_QUOTA command, 8-18
LIST_REMOTE_ID command, 19-6
  ON option, 19-7
LIST_VAR command, 8-11
Literal characters, 4-7
LOAD, 9-3, 11-2
  DC subcommand, 11-9
  helpful PRIMOS commands, 11-9
  INITIALIZE subcommand, 11-9
  invoking, 11-8
  LIBRARY subcommand, 11-9
  LOAD subcommand, 11-9
  MAP subcommand, 11-9
  MODE subcommand, 11-9
  normal linking, 11-9
  order of linking, 11-10
  QUIT subcommand, 11-9
  SAVE subcommand, 11-9
LOAD, BIND subcommand, 11-4
LOAD command, 11-8
LOAD, ED command, 4-22
LOAD, LOAD subcommand, 11-9
Local area networks, 1-3
  see also: LANs
LOCATE, ED command, 4-13
LOG, option of FTR command, 19-19
Logging in, 1-10
  problems, 1-12
  remote, 19-7
  with LAN 300, 1-13
  with NETLINK, 19-10
Logical device numbers, 18-2 to 18-3
Logical device numbers, disks, 19-4
Logical disks, 2-6
LOGIN command, 1-10
  alternate form, 1-13
  ON option, 19-7
Login files, 8-13
Login files, alternate forms, 8-15
LOGIN.CPL, sample file, 8-13
LOGOUT command, 1-14
  with phantoms, 16-3
LON command, 16-3
LONG, option of RDY command, 12-9

M

Magnetic tapes
  see: Tapes
MAGRST command, 18-6
MAGSAV command, 18-5
MAP, BIND subcommand, 11-5
MAP, LOAD subcommand, 11-9
Master File Directories, 2-6
MERGE, option of SORT command, 17-2, 17-5
Merging text files, 17-8
MESSAGE command, 8-16
  ACCEPT option, 8-17
  DEFER option, 8-17
  in Batch jobs, 16-15
  REJECT option, 8-17
  STATUS option, 8-17
Messages, receive states, 8-17
Messages, sending, 8-16
MFDs, 2-6
Mini-command level, 12-8
Mini-commands, list of, 12-9
MINL, option of CMPF command, 17-7
MODE, LOAD subcommand, 11-9
MODE REMOTE_ECHO, option of NETLINK command, 19-12
MODIFY, option of SPOOL command, 4-29
MRGF command, 17-8
  FORCE option, 17-8
  GO subcommand, 17-8
  interactive mode, 17-8
  OUTF option, 17-8
  x subcommand, 17-8
MT, argument to ASSIGN command, 18-2
MT, option of UNASSIGN command, 18-4

N

Name generation, 6-10
Name generation, list of characters, 6-12
NETLINK, 19-9
  @ prompt, 19-10
  C subcommand, 19-10
  example of, 19-10
  HELP subcommand, 19-12

logging in, 19-10
logging out, 19-10
QUIT subcommand, 19-10
when to use, 19-12
NETLINK command, 19-10
MODE REMOTE_ECHO option, 19-12
TO option, 19-11
Network Terminal Service, 1-5
Networks, 1-3, 19-1
LAN, 1-3, 19-1
LAN300, 1-3, 1-5
local system, 19-1
nodenames, 19-2
NTS, 1-5
point-to-point connections, 1-3
PRIMENET, 1-3
PSDN, 19-1
remote systems, 19-1
RINGNET, 1-3
status of, 19-8
wide area, 1-3
NEXT, ED command, 4-12
NFIND, ED command, 4-14
NO_COMOUTPUT, option of JOB command, 16-9
NO_DEFER, option of JOB command, 16-9
Nodenames, 19-2, 19-10
NONE access, 5-27
NO_NOTIFY, option of JOB command, 16-9
NONTAG, option of SORT command, 17-2
NO_QUERY, option of EDIT_ACCESS command, 5-8
NO_QUERY, option of SET_ACCESS command, 5-7, 5-11
NOTIFY, option of JOB command, 16-9, 16-14
NOTIFY, option of SPOOL command, 4-28
NOXOFF, option of TERM command, 1-16
NTS, 1-5
NTS CONNECT command, 1-13
NTTY, option of COMOUTPUT command, 14-3
NULL, CPL function, 15-12
Null string, 15-7

O

O access, 5-23
Object files, 9-2
creating, 10-4
naming, 10-4
Objectnames, 2-1, 2-3
characters in, 2-3
suffixes, 2-3
OFF, option of ABBREV command, 8-8
OFF, option of ECL command, 7-2
OK, prompt, 1-6, 8-2
ON, option of ABBREV command, 8-8
ON, option of ECL command, 7-2
ON, option of LIST_REMOTE_ID command, 19-7
ON, option of LOGIN command, 19-7
On-units, 20-1
default, 20-2
writing, 20-2
OOPS, ED command, 4-19
Operating system, 1-1
see also: PRIMOS
Ordinary pathnames, 2-8
Origin directory, 2-9
OUTF, option of MRGF command, 17-8
Owner access, 5-23

P

P access, 5-23
Packet Switched Data Networks
see: PSDNs
Parent directory, 2-6
Partitions, 2-6
PASSWD command, F-1
Password-protected directories, 2-7
converting to ACLs, F-3
setting access rights, F-2
Passwords, 1-10
assigning or changing, 1-14
computer generated, 1-11
directory, F-1
lifetime, 1-11
see also: Directory passwords
Pathnames, 2-6
absolute, 2-6
directory passwords in, 2-8
full, 2-8
illustration, 2-7
omitting the MFD name, 2-8
ordinary, 2-8
relative, 2-9
shortening, 2-8
specifying, 2-6
uniqueness of, 2-8
PAUSE, option of COMINPUT command, 14-6
PAUSE, option of COMOUTPUT command, 14-3
Peripheral devices, tape drives, 18-1
PHANTOM command, 16-1
Phantom processes, 1-3
Phantoms, 16-1
command file example, 16-4
command input files, 16-2
COMO files, opening, 16-2
CPL programs, 16-2
global variables in, 16-3
initiating other, 16-2
input and output, 16-2
logout, 16-3
status, 16-4
Physical device numbers, 18-2 to 18-3
POINT, ED command, 4-13
Point-to-point connections, 1-3
PRIMENET, 1-3, 19-1
facilities, list of, 19-1
FTS, 19-13
NETLINK facility, 19-9
network status, 19-8
nodenames, 19-10
remote file access, 19-2
remote login, 19-7
virtual circuits, 19-8
PRIMOS, 1-1
command environment, 12-5
command level, 1-7
command line, 1-7
command processor, 1-7
compatibility, 1-3
condition mechanism, 20-1
control characters, 1-9
erase character, 1-8
file system, 2-1
kill character, 1-8
operating system overview, 1-1
processes, 16-1
prompts, 1-6
reserved characters, 1-9
subsystems, 1-7
terminal connections, 1-4
user dialog, 1-5
user interface, 1-2
PRIMOS commands
ABBREV, 8-4
ADD_REMOTE_ID, 19-5
ASSIGN, 18-1
ATTACH, 3-3, F-1
BATCH, 16-16
BATGEN, 16-17
BIND, 11-3
CHANGE_PASSWORD, 1-14
CLOSE, 3-16, 14-2
CMPF, 17-6

CNAME, 3-7
COMINPUT, 14-5
COMOUTPUT, 3-14, 14-2
CONCAT, 17-10
COPY, 3-8, 19-4
CPL, 15-2
CREATE, 3-5
DATE, 14-4
DEFINE_GVAR, 8-11
DELETE, 3-11, 5-14
DELETE_VAR, 8-11
ECL, 7-2
ED, 4-5
EDIT_ACCESS, 5-8, 5-13
essential, 1-10
file system, 3-1
FILMEM, 11-9
FTR, 19-14
HELP, 1-16
in CPL programs, 15-3
INITIALIZE_COMMAND_ENVIRONMENT, 12-9 to 12-10
JOB, 16-7
leading, 3-1
LIST_ACCESS, 5-3, 5-9, 5-15, 5-28
LIST_EPF, 12-9
LIST_LIMITS, 12-7, 12-9
LIST_MINI_COMMANDS, 12-9, 12-12
LIST_PRIORITY_ACCESS, 5-28
LIST_QUOTA, 8-18
LIST_REMOTE_ID, 19-6
LIST_VAR, 8-11
LOAD, 11-8
LOGIN, 1-10, 19-7
LOGOUT, 1-14
LON, 16-3
MESSAGE, 8-16, 16-15
MRGF, 17-8
NETLINK, 19-10
ORIGIN, 3-5
PASSWD, F-1
PHANTOM, 16-1
PROTECT, F-3
RDY, 8-2, 12-9
RELEASE_LEVEL, 3-16, 12-9
REMOVE_REMOTE_ID, 19-7
RESUME, 12-2, 15-2
REVERT_PASSWORD, F-5
SEG, 12-2
SET_ACCESS, 5-6, 5-11 to 5-12, 5-18, 5-20, F-3
SET_DELETE, 3-12
SET_QUOTA, 8-18
SET_VAR, 8-11
SIZE, 8-18
SLIST, 3-6
SORT, 17-1
SPOOL, 4-24
START, 12-3
STATUS DEVICE, 18-3
STATUS DISKS, 19-3
STATUS NETWORK, 19-8
STATUS UNITS, 14-3
STATUS USERS, 16-4
TERM, 1-9, 1-15
UNASSIGN, 18-3
using objectnames in, 2-10
using pathnames in, 2-10
PRINT, ED command, 4-10
Printer environments, 4-27
destinations, 4-27
form names, 4-27
Printing text files, 4-24
at remote sites, 19-16
deferring, 4-27
multiple copies of, 4-26
printing part of a file, 4-28
Priority ACLs, 5-28
LIST_ACCESS command, 5-28
LIST_PRIORITY_ACCESS, 5-28
Priority Batch queues, 16-18
PRIORITY, option of FTR command, 19-16
PRIORITY, option of JOB command, 16-9, 16-11
Processes, 16-1
Program listing files, 10-6
Programming, advanced overview, 9-1
Programming, introductory overview, 9-1
Programs
combining languages in, 10-8
compiling, 9-2, 10-1
CPL, 15-2
creating source code, 9-2
debugging, 9-3, 13-1
invoking, 12-2
linking, 9-2, 11-1
phantom execution, 16-1
restarting, 12-3
running, 9-3, 12-1
Programs: debugging
see: DBG
PROJECT, option of JOB command, 16-9
PROMPT, option of ADD_REMOTE_ID command, 19-5 to 19-6
Prompts, 1-6
changing, 8-1
default, 1-6
error, 8-3
ER!, 1-6
OK, 1-6
long, 8-1
message text, 8-2
NETLINK, 19-10
NTS, 1-13
ready, 8-3
subsystem, 1-6
system, 1-6
warning, 8-3
Protect access, 5-23
PROTECT command, F-2
PROTECT, option of leading command, 3-3, 3-13
Protecting files
from deletion, 3-12
with ACLs, 5-2
Protection codes for password-protected directories, F-3
PSDNs, 19-1

Q

Queue names, Batch system, 16-18
QUEUE, option of JOB command, 16-9, 16-11
QUIT, BIND subcommand, 11-5
QUIT, DBG subcommand, 13-5
QUIT, ED command, 4-22
QUIT, LOAD subcommand, 11-9
QUIT, NETLINK subcommand, 19-10

R

R access, 5-26
R mode, 10-6
RDY command, 8-2, 12-9
list of options, 8-3
Read access, 5-26
Ready prompts, 8-3
READY_BRIEF, option of ECL command, 7-23
Record, in ASCII files, 17-1
REJECT, option of MESSAGE command, 8-17
Relative pathnames, 2-9
RELEASE, option of FTR command, 19-22
RELEASE_LEVEL command, 3-16, 12-9
LONG option, 12-9
Remote file access
adding disks, 19-2
errors, 19-6
remote IDs, 19-5

Remote File Access, 19-2
Remote IDs
  adding, 19-5
  examining, 19-6
  removing, 19-7
Remote login, 19-7
Remote Login, when to use, 19-9
REMOVE_REMOTE_ID command, 19-7
REPORT, option of CMPF command, 17-7
Reserved characters, 1-9
$REST identifier in ACLs, 5-4
RESTART, DBG subcommand, 13-8
RESTART, option of JOB command, 16-9, 16-11
Restarting programs, 12-3
Restoring files from tape, 18-6
RESUME command, 12-2, 15-2
&RETURN, CPL directive, 15-15
Returning to the origin directory, 3-5
RETYPE, ED command, 4-18
REVERT_PASSWORD command, F-5
RINGNET, 1-3, 19-1
Runfiles, 9-2, 11-1
  naming, 12-3
Running programs, 9-3
Running programs interactively, 12-1
RUNOFF, 4-4
Runtime error messages, D-1, 12-5

S

SAVE, ED command, 4-23
SAVE, LOAD subcommand, 11-9
SCR_SITE, option of FTR command, 19-15
SEG command, 12-2
SEG utility, 12-2
Segment directories, 2-2
  symbol, 2-2
Segments
  dynamic, 12-7
  limits, 12-7
  static, 12-7
SET_ACCESS command, 5-6, 5-11 to 5-12, 5-18, F-3
  LIKE option, 5-20
  NO_QUERY option, 5-11
SET_DELETE command, 3-12
SET_QUOTA command, 8-18
Setting terminal characteristics, 1-15
SET_VAR command, 8-11
&SET_VAR, CPL directive, 15-7
&SEVERITY, CPL directive, 15-16
Sites, 19-14
SIZE command, 8-18
SLIST command, 3-6
SORT command, 17-1
  key fields, 17-3
  key specifications, 17-4
  MERGE option, 17-5
  mergesort example, 17-5
  options, list of, 17-2
Sort keys, 17-3
Sorting files, 17-1
Source code, 9-2
SOURCE, DBG subcommand, 13-5
  arguments, 13-5
  line numbers, 13-7
Source files, 10-1
Source Level Debugger
  see: DBG
SOURCE_USER, option of FTR command, 19-20
SPACE, option of SORT command, 17-2
Specific ACLs, 5-2
SPOOL command, 4-24
  ATTRIBUTE option, 4-27
  CANCEL option, 4-26
  canceling requests, 4-26
  COPIES option, 4-26
  DEFER option, 4-27
  FROM option, 4-28
  LIST option, 4-25
  MODIFY option, 4-29
  modifying options, 4-29
  multiple options, 4-28
  NOTIFY option, 4-28
  TO option, 4-28
SPOOL queue, 4-25
  illustration, 4-25
  status of, 4-25
Spooling
  see: Printing
SRC_NTFY, option of FTR command, 19-20
STACK, option of ECL command, 7-15
START command, 12-3
START, option of COMINPUT command, 14-6
Static runfiles, 11-2
  converting to EPFs, 11-2
Static segments, 12-7
  see also: Static runfiles
Static-mode programs
  converting to EPFs, 11-2
  restarting, 12-4
STATUS DEVICE command, 18-3
STATUS DISKS command, 19-3
STATUS NETWORK command, 19-8
STATUS, option of BATCH command, 16-16
STATUS, option of BATGEN command, 16-17
STATUS, option of FTR command, 19-17
STATUS, option of JOB command, 16-14
STATUS, option of MESSAGE command, 8-17
STATUS UNITS command, 14-3
STATUS USERS command, 16-4
STEP, DBG subcommand, 13-9
STICK, option of ECL command, 7-14
Subdirectories, 2-6
Subsystems, 1-7
Suffixes, 2-3
Suffixes, compiler name, 10-3
Syntax suppression, 6-13
System Administrator, 1-10
System level, 1-7
System names
  see: Node names
System prompts, 1-7
Systemnames, 19-2, 19-4, 19-14

T

TAG, option of SORT command, 17-2
Tape drives
  assigning, 18-1
  releasing, 18-3
  unassigning, 18-3
TERM command, 1-9, 1-15
  BREAK option, 1-15
  DISPLAY option, 1-16
  ERASE option, 1-15
  KILL option, 1-15
  NOXOFF option, 1-16
  XOFF option, 1-15
Terminal sessions, recording, 3-14
Terminals, 1-4
  communication rate, 1-5
  connections, 1-5
  dialup, 1-5
  input buffer, 1-6
  LAN Terminal Server, 1-5
  NTS, 1-5
Text editors, 4-2
  ED, 4-2
  editing, 4-2
  editing window, 4-2
  EMACS, 4-2
  line-oriented, 4-2
  saving, 4-2

screen-oriented, 4-2
Text files, 4-1
Text formatter, 4-4
Timeslice, 16-19
TO, option of NETLINK command, 19-11
TO, option of SPOOL command, 4-28
TOP, ED command, 4-11
Top-level directories, 2-6
Transferring files with FTS, 19-13
Treewalking, 6-8
options, 6-10
&TTY, CPL directive, 15-14
TTY, option of COMINPUT command, 14-6
TTY, option of COMOUTPUT command, 14-3
Type-ahead, 1-6

U

U access, 5-26
UNASSIGN command, 18-3
ALIAS option, 18-4
MT argument, 18-4
UNLOAD option, 18-4
Unassigning tape drives, 18-3
UNLOAD, ED command, 4-20
UNLOAD, option of UNASSIGN command, 18-4
Use access, 5-26
User interface, 1-2
User's dialog with PRIMOS, 1-5

V

V mode, 10-5
Variables
in abbreviations, 8-7
in CPL programs, 15-4
Variables, global
see: Global variables
Virtual circuits, 19-8

W

W access, 5-27
WAIT, option of ASSIGN command, 18-2
WALK_FROM, treewalking option, 6-10
WALK_TO, treewalking option, 6-10
Warning prompts, 8-3
WARNING_BRIEF, option of ECL command, 7-23
WATCH, DBG subcommand, 13-10
WHERE, ED command, 4-12
Wide area networks, 1-3
Wildcards, 6-2
leading command, 6-6
list of characters, 6-3
options, 6-4
with iteration lists, 6-4
WILD_TAIL, option of ECL command, 7-17
Write access, 5-27

X

X access, 5-27
x, subcommand of MRGF, 17-8
XOFF, option of TERM command, 1-15
XREF, compiler option, 10-7

Surveys

Reader Response Form

PRIMOS User's Guide

DOC4130-5LA

Your feedback will help us continue to improve the quality, accuracy, and organization of our user publications.

1. How do you rate this document for overall usefulness?

   ☐ *excellent* ☐ *very good* ☐ *good* ☐ *fair* ☐ *poor*

2. What features of this manual did you find most useful?

3. What faults or errors in this manual gave you problems?

4. How does this manual compare to equivalent manuals produced by other computer companies?

☐ *Much better*	☐ *Slightly better*	☐ *About the same*
☐ *Much worse*	☐ *Slightly worse*	☐ *Can't judge*

5. Which other companies' manuals have you read?

Name:

Position:

Company:

Address:

Postal Code:

NO POSTAGE
NECESSARY
IF MAILED
IN THE
UNITED STATES

First Class Permit #531 Natick, Massachusetts 01760

BUSINESS REPLY MAIL

Postage will be paid by:

Prime™

Attention: Technical Publications
Bldg 10
Prime Park, Natick, Ma. 01760